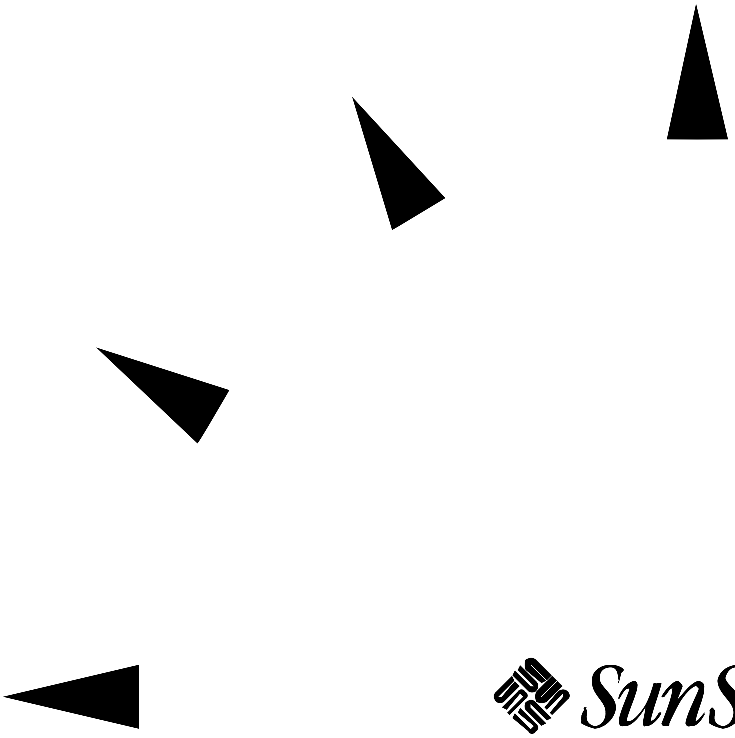
SunOS Reference Manual



Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

Preface

OVERVIEW

A man page is provided for both the naive user, and sophisticated user who is familiar with the SunOS operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following contains a brief description of each section in the man pages and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.

- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.

- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.

- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume.

- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character set tables, etc.
- Section 7 describes various special files that refer to specific hardware peripherals, and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver–Kernel Interface (DKI).
- Section 9E describes the DDI / DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the intro pages for more information and detail about each section, and **man**(1) for more information about man pages in general.

NAME

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Literal characters (commands and options) are in **bold** font and variables (arguments, parameters and substitution characters) are in *italic* font. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ]	The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument *must* be specified.
. . .	Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, '*filename* . . .'.
|	Separator. Only one of the arguments separated by this character can be specified at time.

PROTOCOL

This section occurs only in subsection 3R to indicate the protocol description file. The protocol specification pathname is always listed in **bold** font.

AVAILABILITY

This section briefly states any limitations on the availabilty of the command. These limitations could be hardware or software specific.

A specification of a class of hardware platform, such as **x86** or **SPARC**, denotes that the command or interface is applicable for the hardware platform specified.

In Section 1 and Section 1M, **AVAILABILITY** indicates which package contains the command being described on the manual page. In order to use the command, the specified package must have been installed with the operating system. If the package was not installed, see **pkgadd**(1) for information on how to upgrade.

MT-LEVEL

This section lists the **MT-LEVEL** of the library functions described in the Section 3 manual pages. The **MT-LEVEL** defines the libraries' ability to support threads. See **Intro**(3) for more information.

DESCRIPTION

This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

IOCTLS

This section appears on pages in Section 7 only. Only the device class which supplies appropriate parameters to the **ioctls**(2) system call is called **ioctls** and generates its own heading. IOCTLS for a specific device are listed alphabetically (on the man page for that specific device). IOCTLS are used for a particular class of devices all which have an **io** ending, such as **mtio**(7).

OPTIONS

This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

RETURN VALUES

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared as **void** do not return values, so they are not discussed in RETURN VALUES.

ERRORS

On failure, most functions place an error code in the global variable **errno** indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE

This section is provided as a *guidance* on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:

Commands
Modifiers
Variables
Expressions
Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as

example%

or if the user must be super-user,

example#

Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.

ENVIRONMENT

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

FILES

This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications.

DIAGNOSTICS

This section lists diagnostic messages with a brief explanation of the condition causing the error. Messages appear in **bold** font with the exception of variables, which are in *italic* font.

WARNINGS

This section lists warnings about special conditions which could seriously affect your working conditions — this is not a list of diagnostics.

NOTES

This section lists additional information that does not belong anywhere else on the page. It takes the form of an *aside* to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and wherever possible suggests workarounds.

NAME

Intro, intro – introduction to commands and application programs

AVAILABILITY

This section indicates which package contains the commands being described on this page. To be able to use the command, the indicated package must have been installed with the operating system. For information on how to add a package see **pkgadd**(1).

DESCRIPTION

This section describes, in alphabetical order, commands available with this operating system.

Pages of special interest are categorized as follows:

1B	Commands found only in the *SunOS/BSD Compatibility Package*. Refer to the *Solaris Source Compatibility Guide* for more information.
1C	Commands for communicating with other systems.
1F	Commands associated with *Form and Menu Language Interpreter* (FMLI).
1S	Commands specific to the SunOS system.

OTHER SECTIONS

See these sections of the *man Pages(1M): System Administration Commands* for more information.

- Section 1M in this manual for system maintenance commands.
- Section 4 of this manual for information on file formats.
- Section 5 of this manual for descriptions of publicly available files and miscellaneous information pages.
- Section 6 in this manual for computer demonstrations.

For tutorial information about these commands and procedures, see:

- *Solaris Advanced User's Guide*
- *Programming Utilities Guide*

Manual Page Command Syntax

Unless otherwise noted, commands described in the **SYNOPSIS** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [–*option*...] [*cmdarg*...]
where:

[ ]	Surround an *option* or *cmdarg* that is not required.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "–".) *noargletter*... or, *argletter optarg*[**,**...]

noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "–" (Rule 5, below).
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or separated by a tab or space character and quoted (Rule 8, below).
cmdarg	Path name (or other command argument) *not* beginning with "–", or "–" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. **getopts**(1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.
2. Command names must include only lower-case letters and digits.
3. Option names (*option* above) must be one character long.
4. All options must be preceded by "–".
5. Options with no arguments may be grouped after a single "–".
6. The first option-argument (*optarg* above) following an option must be preceded by a tab or space character.
7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by tab or space character and quoted (**–o xxx,z,yy** or **–o** "**xxx z yy**").
9. All options must precede operands (*cmdarg* above) on the command line.
10. "––" may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. "–" preceded and followed by a space character should only be used to mean standard input.

SEE ALSO

getopts(1), **wait**(1), **exit**(2), **getopt**(3C), **wait**(3B)

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of “normal” termination) one supplied by the program [see **wait**(3B) and **exit**(2)]. The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, or bad or inaccessible data. It is called variously “exit code”, “exit status”, or “return code”, and is described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

Name	**Appears on Page**	**Description**
acctcom	**acctcom**(1)	search and print process accounting files
adb	**adb**(1)	general-purpose debugger
addbib	**addbib**(1)	create or extend a bibliographic database
admin	**sccs-admin**(1)	create and administer SCCS history files
aedplot	**plot**(1B)	graphics filters for various plotters
alias	**alias**(1)	shell built-in functions to create your own pseudonym or shorthand for a command or series of commands
apropos	**apropos**(1)	locate commands by keyword lookup
ar	**ar**(1)	maintain portable archive or library
arch	**arch**(1B)	display the architecture of the current host
as	**as**(1)	assembler
at	**at**(1)	execute commands at a later time
atoplot	**plot**(1B)	graphics filters for various plotters
atq	**atq**(1)	display the jobs queued to run at specified times
atrm	**atrm**(1)	remove jobs spooled by at or batch
audioconvert	**audioconvert**(1)	convert audio file formats
audioplay	**audioplay**(1)	play audio files
audiorecord	**audiorecord**(1)	record an audio file
awk	**awk**(1)	pattern scanning and processing language
banner	**banner**(1)	make posters
basename	**basename**(1)	deliver portions of pathnames
basename	**basename**(1B)	display portions of pathnames
batch	**at**(1)	execute commands at a later time
bc	**bc**(1)	arbitrary precision arithmetic language
bdiff	**bdiff**(1)	big diff
bg	**jobs**(1)	shell built-in functions to control process execution
bgplot	**plot**(1B)	graphics filters for various plotters

biff	**biff**(1B)	give notice of incoming mail messages
break	**break**(1)	shell built-in functions to escape from or advance within a controlling while, for, foreach, or until loop
cal	**cal**(1)	display a calendar
calendar	**calendar**(1)	reminder service
cancel	**lp**(1)	send / cancel requests to an LP print service
case	**case**(1)	shell built-in functions to choose from among a list of actions
cat	**cat**(1)	concatenate and display files
cc	**cc**(1B)	C compiler
cd	**cd**(1)	shell built-in functions to change the current working directory
cdc	**sccs-cdc**(1)	change the delta commentary of an SCCS delta
chdir	**cd**(1)	shell built-in functions to change the current working directory
checkeq	**eqn**(1)	typeset mathematics test
checknr	**checknr**(1)	check nroff and troff input files; report possible errors
chgrp	**chgrp**(1)	change the group ownership of a file
chkey	**chkey**(1)	change user's secure RPC key pair
chmod	**chmod**(1)	change the permissions mode of a file
chown	**chown**(1)	change owner of file
chown	**chown**(1B)	change owner
ckdate	**ckdate**(1)	prompts for and validates a date
ckgid	**ckgid**(1)	prompts for and validates a group id
ckint	**ckint**(1)	display a prompt; verify and return an integer value
ckitem	**ckitem**(1)	build a menu; prompt for and return a menu item
ckkeywd	**ckkeywd**(1)	prompts for and validates a keyword
ckpath	**ckpath**(1)	display a prompt; verify and return a pathname
ckrange	**ckrange**(1)	prompts for and validates an integer
ckstr	**ckstr**(1)	display a prompt; verify and return a string answer
cktime	**cktime**(1)	display a prompt; verify and return a time of day
ckuid	**ckuid**(1)	prompts for and validates a user ID
ckyorn	**ckyorn**(1)	prompts for and validates yes / no
clear	**clear**(1)	clear the terminal screen
cmp	**cmp**(1)	compare two files
cocheck	**coproc**(1F)	communicate with a process
cocreate	**coproc**(1F)	communicate with a process
codestroy	**coproc**(1F)	communicate with a process
cof2elf	**cof2elf**(1)	COFF to ELF object file translation

col	**col**(1)	reverse line-feeds filter
comb	**sccs-comb**(1)	combine SCCS deltas
comm	**comm**(1)	select or reject lines common to two sorted files
compress	**compress**(1)	compress, uncompress files or display expanded files
continue	**break**(1)	shell built-in functions to escape from or advance within a controlling while, for, foreach, or until loop
coproc	**coproc**(1F)	communicate with a process
coreceive	**coproc**(1F)	communicate with a process
cosend	**coproc**(1F)	communicate with a process
cp	**cp**(1)	copy files
cpio	**cpio**(1)	copy file archives in and out
cpp	**cpp**(1)	the C language preprocessor
crontab	**crontab**(1)	user crontab file
crtplot	**plot**(1B)	graphics filters for various plotters
crypt	**crypt**(1)	encode or decode a file
csh	**csh**(1)	shell command interpreter with a C-like syntax
csplit	**csplit**(1)	split a file with respect to a given context
ct	**ct**(1C)	spawn login to a remote terminal
ctags	**ctags**(1)	create a tags file for use with ex and vi
cu	**cu**(1C)	call another UNIX system
cut	**cut**(1)	cut out selected fields of each line of a file
date	**date**(1)	print and set the date
dc	**dc**(1)	desk calculator
delta	**sccs-delta**(1)	make a delta to an SCCS file
deroff	**deroff**(1)	remove nroff / troff, tbl, and eqn constructs
df	**df**(1B)	display status of disk space on file systems
diff3	**diff3**(1)	3-way differential file comparison
diff	**diff**(1)	display line-by-line differences between pairs of text files
diffmk	**diffmk**(1)	mark differences between versions of a troff input file
dircmp	**dircmp**(1)	directory comparison
dirname	**basename**(1)	deliver portions of pathnames
dirs	**cd**(1)	shell built-in functions to change the current working directory
dis	**dis**(1)	object code disassembler
disable	**enable**(1)	enable / disable LP printers
dispgid	**dispgid**(1)	displays a list of all valid group names
dispuid	**dispuid**(1)	displays a list of all valid user names
dos2unix	**dos2unix**(1)	convert text file from DOS format to ISO format
download	**download**(1)	host resident PostScript font downloader

dpost	**dpost**(1)	troff postprocessor for PostScript printers
du	**du**(1B)	display the number of disk blocks used per directory or file
dumbplot	**plot**(1B)	graphics filters for various plotters
dump	**dump**(1)	dump selected parts of an object file
dumpcs	**dumpcs**(1)	show codeset table for the current locale
dumpkeys	**loadkeys**(1)	load and dump keyboard translation tables
echo	**echo**(1)	echo arguments
echo	**echo**(1B)	echo arguments to standard output
echo	**echo**(1F)	put string on virtual output
ed	**ed**(1)	text editor
edit	**edit**(1)	text editor (variant of ex for casual users)
egrep	**egrep**(1)	search a file for a pattern using full regular expressions
eject	**eject**(1)	eject media such as CD-ROM and floppy from drive
enable	**enable**(1)	enable / disable LP printers
env	**env**(1)	obtain or alter environment variables for command execution
eqn	**eqn**(1)	typeset mathematics test
errange	**ckrange**(1)	prompts for and validates an integer
errdate	**ckdate**(1)	prompts for and validates a date
errgid	**ckgid**(1)	prompts for and validates a group id
errint	**ckint**(1)	display a prompt; verify and return an integer value
erritem	**ckitem**(1)	build a menu; prompt for and return a menu item
error	**error**(1)	insert compiler error messages at right source lines
errpath	**ckpath**(1)	display a prompt; verify and return a pathname
errstr	**ckstr**(1)	display a prompt; verify and return a string answer
errtime	**cktime**(1)	display a prompt; verify and return a time of day
erruid	**ckuid**(1)	prompts for and validates a user ID
erryorn	**ckyorn**(1)	prompts for and validates yes / no
eval	**exec**(1)	shell built-in functions to execute other commands
ex	**ex**(1)	text editor
exec	**exec**(1)	shell built-in functions to execute other commands
exit	**exit**(1)	shell built-in functions to enable the execution of the shell to advance beyond its sequence of steps

expand	**expand**(1)	expand TAB characters to SPACE characters, and vice versa
export	**set**(1)	shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents
exportfs	**exportfs**(1B)	translates exportfs options to share/unshare commands
expr	**expr**(1)	evaluate arguments as an expression
expr	**expr**(1B)	evaluate arguments as a logical, arithmetic, or string expression
exstr	**exstr**(1)	extract strings from source files
face	**face**(1)	executable for the Framed Access Command Environment Interface
factor	**factor**(1)	obtain the prime factors of a number
false	**true**(1)	provide truth values
fastboot	**fastboot**(1B)	reboot/halt the system without checking the disks
fasthalt	**fastboot**(1B)	reboot/halt the system without checking the disks
fc	**history**(1)	shell built-in functions to re-use previous command-lines from the current shell
fdformat	**fdformat**(1)	format floppy diskette
fg	**jobs**(1)	shell built-in functions to control process execution
fgrep	**fgrep**(1)	search a file for a character string
file	**file**(1)	determine file type
file	**file**(1B)	determine the type of a file by examining its contents
find	**find**(1)	find files
finger	**finger**(1)	display information about local and remote users
fmlcut	**fmlcut**(1F)	cut out selected fields of each line of a file
fmlexpr	**fmlexpr**(1F)	evaluate arguments as an expression
fmlgrep	**fmlgrep**(1F)	search a file for a pattern
fmli	**fmli**(1)	invoke FMLI
fmt	**fmt**(1)	simple text formatters
fmtmsg	**fmtmsg**(1)	display a message on stderr or system console
fold	**fold**(1)	fold long lines
for	**for**(1)	shell built-in functions to repeatedly execute action(s) for a selected number of times
foreach	**for**(1)	shell built-in functions to repeatedly execute action(s) for a selected number of times
from	**from**(1B)	display the sender and date of newly-arrived mail messages
ftp	**ftp**(1)	file transfer program

function	**function**(1)	shell built-in command to define a function which is usable within this shell
gcore	**gcore**(1)	get core images of running processes
gencat	**gencat**(1)	generate a formatted message catalog
get	**sccs-get**(1)	retrieve a version of an SCCS file
getfrm	**getfrm**(1F)	returns the current frameID number
getitems	**getitems**(1F)	returns a list of currently marked menu items
getopt	**getopt**(1)	parse command options
getoptcvt	**getoptcvt**(1)	convert to getopts to parse command options
getopts	**getopts**(1)	shell built-in function to parse command-line options
gettext	**gettext**(1)	retrieve text string from message database
gettxt	**gettxt**(1)	retrieve a text string from a message database
gigiplot	**plot**(1B)	graphics filters for various plotters
glob	**glob**(1)	shell built-in function to expand a word list
goto	**exit**(1)	shell built-in functions to enable the execution of the shell to advance beyond its sequence of steps
gprof	**gprof**(1)	display call-graph profile data
graph	**graph**(1)	draw a graph
grep	**grep**(1)	search a file for a pattern
groups	**groups**(1)	print group membership of user
groups	**groups**(1B)	display a user’s group memberships
grpck	**grpck**(1B)	check group database entries
hash	**hash**(1)	shell built-in functions to evaluate the internal hash table of the contents of directories
hashstat	**hash**(1)	shell built-in functions to evaluate the internal hash table of the contents of directories
head	**head**(1)	display first few lines of files
help	**sccs-help**(1)	ask for help regarding SCCS error or warning messages
helpdate	**ckdate**(1)	prompts for and validates a date
helpgid	**ckgid**(1)	prompts for and validates a group id
helpint	**ckint**(1)	display a prompt; verify and return an integer value
helpitem	**ckitem**(1)	build a menu; prompt for and return a menu item
helppath	**ckpath**(1)	display a prompt; verify and return a pathname
helprange	**ckrange**(1)	prompts for and validates an integer
helpstr	**ckstr**(1)	display a prompt; verify and return a string answer

helptime	**cktime**(1)	display a prompt; verify and return a time of day
helpuid	**ckuid**(1)	prompts for and validates a user ID
helpyorn	**ckyorn**(1)	prompts for and validates yes/no
history	**history**(1)	shell built-in functions to re-use previous command-lines from the current shell
hostid	**hostid**(1B)	print the numeric identifier of the current host
hostname	**hostname**(1B)	set or print name of current host system
hp7221plot	**plot**(1B)	graphics filters for various plotters
hpplot	**plot**(1B)	graphics filters for various plotters
i286	**machid**(1)	get processor type truth value
i386	**machid**(1)	get processor type truth value
i486	**machid**(1)	get processor type truth value
i860	**machid**(1)	get processor type truth value
iAPX286	**machid**(1)	get processor type truth value
iconv	**iconv**(1)	code set conversion utility
if	**if**(1)	shell built-in functions to evaluate expression(s) or to make execution of actions dependent upon the evaluation of expression(s)
implot	**plot**(1B)	graphics filters for various plotters
indicator	**indicator**(1F)	display application specific alarms and/or the "working" indicator
indxbib	**indxbib**(1)	create an inverted index to a bibliographic database
install	**install**(1B)	install files
ipcrm	**ipcrm**(1)	remove a message queue, semaphore set, or shared memory ID
ipcs	**ipcs**(1)	report inter-process communication facilities status
jobs	**jobs**(1)	shell built-in functions to control process execution
join	**join**(1)	relational database operator
jsh	**sh**(1)	shell: the standard shell, and job control shell -- command interpreters
kbd	**kbd**(1)	manipulate the state of keyboard or display the type of keyboard
kdestroy	**kdestroy**(1)	destroy Kerberos tickets
kerberos	**kerberos**(1)	introduction to the Kerberos system
keylogin	**keylogin**(1)	decrypt and store secret key with keyserv
keylogout	**keylogout**(1)	delete stored secret key with keyserv
kill	**kill**(1)	terminate a process by default
kinit	**kinit**(1)	Kerberos login utility
klist	**klist**(1)	list currently held Kerberos tickets

ksh	**ksh**(1)	KornShell, a standard/restricted command and programming language
ksrvtgt	**ksrvtgt**(1)	fetch and store Kerberos ticket-granting ticket using a service key
last	**last**(1)	display login and logout information about users and terminals
lastcomm	**lastcomm**(1)	display the last commands executed, in reverse order
ld	**ld**(1)	link editor for object files
ld	**ld**(1B)	link editor, dynamic link editor
ldd	**ldd**(1)	list dynamic dependencies of executable files or shared objects
let	**let**(1)	shell built-in function to evaluate one or more arithmetic expressions
lex	**lex**(1)	lexical analysis program generator
limit	**limit**(1)	shell built-in functions to set/get limitations on the system resources available to the current shell and it's descendents.
line	**line**(1)	read one line
listusers	**listusers**(1)	list user login information
ln	**ln**(1)	make hard or symbolic links to files
ln	**ln**(1B)	make hard or symbolic links to files
loadfont	**loadfont**(1)	display or change font information in the RAM of the video card on an x86 system in text mode
loadkeys	**loadkeys**(1)	load and dump keyboard translation tables
logger	**logger**(1)	add entries to the system log
logger	**logger**(1B)	add entries to the system log
login	**login**(1)	sign on to the system
logname	**logname**(1)	get the name of the user running the process
logout	**logout**(1)	shell built-in function to exit from a login session
longline	**readfile**(1F)	reads file, gets longest line
look	**look**(1)	find words in the system dictionary or lines in a sorted list
lookbib	**lookbib**(1)	find references in a bibliographic database
lorder	**lorder**(1)	find ordering relation for an object or library archive
lp	**lp**(1)	send/cancel requests to an LP print service
lpc	**lpc**(1B)	line printer control program
lpq	**lpq**(1B)	display the queue of printer jobs
lpr	**lpr**(1B)	send a job to the printer
lprm	**lprm**(1B)	remove jobs from the printer queue

lpstat	**lpstat**(1)	print information about the status of the LP print service
lptest	**lptest**(1B)	generate lineprinter ripple pattern
ls	**ls**(1)	list contents of directory
ls	**ls**(1B)	list the contents of a directory
m4	**m4**(1)	macro processor
mach	**mach**(1B)	display the processor type of the current host
machid	**machid**(1)	get processor type truth value
Mail	**mailx**(1)	interactive message processing system
mail	**mail**(1)	read mail or send mail to users
mail	**mailx**(1)	interactive message processing system
mailstats	**mailstats**(1)	print statistics collected by sendmail
mailx	**mailx**(1)	interactive message processing system
make	**make**(1S)	maintain, update, and regenerate related programs and files
man	**man**(1)	find and display reference manual pages
mconnect	**mconnect**(1)	connect to SMTP mail server socket
mcs	**mcs**(1)	manipulate the comment section of an object file
mesg	**mesg**(1)	permit or deny messages
message	**message**(1F)	puts its arguments on FMLI message line
mkdir	**mkdir**(1)	make directories
mkmsgs	**mkmsgs**(1)	create message files for use by gettxt
mkstr	**mkstr**(1B)	create an error message file by massaging C source files
more	**more**(1)	browse or page through a text file
msgfmt	**msgfmt**(1)	create a message object from a message file
mt	**mt**(1)	magnetic tape control
mv	**mv**(1)	move files
nawk	**nawk**(1)	pattern scanning and processing language
neqn	**eqn**(1)	typeset mathematics test
newaliases	**newaliases**(1)	rebuild the data base for the mail aliases file
newform	**newform**(1)	change the format of a text file
newgrp	**newgrp**(1)	shell built-in function to allow new group permissions to the user
news	**news**(1)	print news items
nice	**nice**(1)	run a command at low priority
NIS+	**nis+**(1)	a new version of the network information name service
nis+	**nis+**(1)	a new version of the network information name service
nis	**nis+**(1)	a new version of the network information name service
niscat	**niscat**(1)	display NIS+ tables and objects

nischgrp	**nischgrp**(1)	change the group owner of a NIS+ object
nischmod	**nischmod**(1)	change access rights on a NIS+ object
nischown	**nischown**(1)	change the owner of a NIS+ object
nischttl	**nischttl**(1)	change the time to live value of a NIS+ object
nisdefaults	**nisdefaults**(1)	display NIS+ default values
niserror	**niserror**(1)	display NIS+ error messages
nisgrep	**nismatch**(1)	utilities for searching NIS+ tables
nisgrpadm	**nisgrpadm**(1)	NIS+ group administration command
nisln	**nisln**(1)	symbolically link NIS+ objects
nisls	**nisls**(1)	list the contents of a NIS+ directory
nismatch	**nismatch**(1)	utilities for searching NIS+ tables
nismkdir	**nismkdir**(1)	create NIS+ directories
nispasswd	**nispasswd**(1)	change NIS+ password information
nisrm	**nisrm**(1)	remove NIS+ objects from the namespace
nisrmdir	**nisrmdir**(1)	remove NIS+ directories
nistbladm	**nistbladm**(1)	NIS+ table administration command
nistest	**nistest**(1)	return the state of the NIS+ namespace using a conditional expression
nl	**nl**(1)	line numbering filter
nm	**nm**(1)	print name list of an object file
nohup	**nohup**(1)	run a command immune to hangups and quits
notify	**jobs**(1)	shell built-in functions to control process execution
nroff	**nroff**(1)	format documents for display or line-printer
od	**od**(1)	octal dump
on	**on**(1)	execute a command on a remote system, but with the local environment
onintr	**trap**(1)	shell built-in functions to respond to (hardware) signals
pack	**pack**(1)	compress and expand files
page	**more**(1)	browse or page through a text file
pagesize	**pagesize**(1B)	display the size of a page of memory
passwd	**passwd**(1)	change login password and password attributes
paste	**paste**(1)	merge same lines of several files or subsequent lines of one file
pathconv	**pathconv**(1F)	search FMLI criteria for filename
pcat	**pack**(1)	compress and expand files
pcmapkeys	**pcmapkeys**(1)	set keyboard extended map and scancode translation for the PC console in text mode
pdp11	**machid**(1)	get processor type truth value
pg	**pg**(1)	files perusal filter for CRTs
pkginfo	**pkginfo**(1)	display software package information
pkgmk	**pkgmk**(1)	produce an installable package
pkgparam	**pkgparam**(1)	displays package parameter values

pkgproto	**pkgproto**(1)	generate prototype file entries for input to pkgmk command
pkgtrans	**pkgtrans**(1)	translate package format
plot	**plot**(1B)	graphics filters for various plotters
plottoa	**plot**(1B)	graphics filters for various plotters
popd	**cd**(1)	shell built-in functions to change the current working directory
postdaisy	**postdaisy**(1)	PostScript translator for Diablo 630 daisy-wheel files
postdmd	**postdmd**(1)	PostScript translator for DMD bitmap files
postio	**postio**(1)	serial interface for PostScript printers
postmd	**postmd**(1)	matrix display program for PostScript printers
postplot	**postplot**(1)	PostScript translator for plot(4) graphics files
postprint	**postprint**(1)	PostScript translator for text files
postreverse	**postreverse**(1)	reverse the page order in a PostScript file
posttek	**posttek**(1)	PostScript translator for Tektronix 4014 files
pr	**pr**(1)	print files
print	**print**(1)	shell built-in function to output characters to the screen or window
printenv	**printenv**(1B)	display environment variables currently set
printf	**printf**(1)	print formatted output
priocntl	**priocntl**(1)	display or set scheduling parameters of specified process(es)
prof	**prof**(1)	display profile data
prs	**sccs-prs**(1)	display selected portions of an SCCS history
prt	**sccs-prt**(1)	display delta table information from an SCCS file
ps	**ps**(1)	report process status
ps	**ps**(1B)	display the status of current processes
pushd	**cd**(1)	shell built-in functions to change the current working directory
pwd	**pwd**(1)	working directory name
rcp	**rcp**(1)	remote file copy
rdist	**rdist**(1)	remote file distribution program
read	**read**(1)	shell built-in function to receive from standard input (keyboard)
readfile	**readfile**(1F)	reads file, gets longest line
readonly	**readonly**(1)	shell built-in function to protect the value of the given variable from reassignment
red	**ed**(1)	text editor
refer	**refer**(1)	expand and insert references from a bibliographic database
regcmp	**regcmp**(1)	regular expression compile
regex	**regex**(1F)	match patterns against a string

rehash	**hash**(1)	shell built-in functions to evaluate the internal hash table of the contents of directories
reinit	**reinit**(1F)	runs an initialization file
remote_shell	**rsh**(1)	remote shell
remsh	**rsh**(1)	remote shell
renice	**renice**(1B)	alter priority of running processes
repeat	**repeat**(1)	shell built-in function to execute a command more than once
reset	**reset**(1F)	reset the current form field to its default values
reset	**tset**(1B)	establish or restore terminal characteristics
return	**exit**(1)	shell built-in functions to enable the execution of the shell to advance beyond its sequence of steps
rksh	**ksh**(1)	KornShell, a standard / restricted command and programming language
rlogin	**rlogin**(1)	remote login
rm	**rm**(1)	remove files or directories
rmail	**mail**(1)	read mail or send mail to users
rmdel	**sccs-rmdel**(1)	remove a delta from an SCCS file
rmdir	**rm**(1)	remove files or directories
roffbib	**roffbib**(1)	format and print a bibliographic database
rpcgen	**rpcgen**(1)	an RPC protocol compiler
rsh	**rsh**(1)	remote shell
run	**run**(1F)	run an executable
rup	**rup**(1)	show host status of remote machines (RPC version)
rup	**rup**(1C)	show host status of remote machines (RPC version)
ruptime	**ruptime**(1)	show host status of local machines
rusage	**rusage**(1B)	print resource usage for a command
rusers	**rusers**(1)	who’s logged in on remote machines
rwho	**rwho**(1)	who’s logged in on local machines
sact	**sccs-sact**(1)	show editing activity status of an SCCS file
sag	**sag**(1)	system activity graph
sar	**sar**(1)	system activity reporter
sccs-admin	**sccs-admin**(1)	create and administer SCCS history files
sccs-cdc	**sccs-cdc**(1)	change the delta commentary of an SCCS delta
sccs-comb	**sccs-comb**(1)	combine SCCS deltas
sccs-delta	**sccs-delta**(1)	make a delta to an SCCS file
sccs-get	**sccs-get**(1)	retrieve a version of an SCCS file
sccs-help	**sccs-help**(1)	ask for help regarding SCCS error or warning messages
sccs-prs	**sccs-prs**(1)	display selected portions of an SCCS history

sccs-prt	**sccs-prt**(1)	display delta table information from an SCCS file
sccs-rmdel	**sccs-rmdel**(1)	remove a delta from an SCCS file
sccs-sact	**sccs-sact**(1)	show editing activity status of an SCCS file
sccs-sccsdiff	**sccs-sccsdiff**(1)	compare two versions of an SCCS file
sccs-unget	**sccs-unget**(1)	undo a previous get of an SCCS file
sccs-val	**sccs-val**(1)	validate an SCCS file
sccs	**sccs**(1)	front end for the Source Code Control System (SCCS)
sccsdiff	**sccs-sccsdiff**(1)	compare two versions of an SCCS file
script	**script**(1)	make record of a terminal session
sdiff	**sdiff**(1)	print differences between two files side-by-side
sed	**sed**(1)	stream editor
select	**case**(1)	shell built-in functions to choose from among a list of actions
set	**set**(1)	shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents
set	**set**(1F)	set and unset local or global environment variables
setcolor	**setcolor**(1F)	redefine or create a color
setenv	**set**(1)	shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents
sh	**sh**(1)	shell: the standard shell, and job control shell -- command interpreters
shell	**shell**(1F)	run a command using shell
shell_builtins	**shell_builtins**(1)	shell command interpreter built-in functions
shift	**shift**(1)	shell built-in function to traverse either a shell's argument list or a list of field-separated words
shutdown	**shutdown**(1B)	close down the system at a given time
size	**size**(1)	print section sizes in bytes of object files
sleep	**sleep**(1)	suspend execution for an interval
soelim	**soelim**(1)	resolve and eliminate .so requests from nroff or troff input
sort	**sort**(1)	sort and / or merge files
sortbib	**sortbib**(1)	sort a bibliographic database
source	**exec**(1)	shell built-in functions to execute other commands
sparc	**machid**(1)	get processor type truth value
spell	**spell**(1)	find spelling errors
spline	**spline**(1)	interpolate smooth curve
split	**split**(1)	split a file into pieces

srchtxt	**srchtxt**(1)	display contents of, or search for a text string in, message data bases
stop	**jobs**(1)	shell built-in functions to control process execution
strchg	**strchg**(1)	change or query stream configuration
strconf	**strchg**(1)	change or query stream configuration
strings	**strings**(1)	find printable strings in an object or binary file
strip	**strip**(1)	strip symbol table, debugging and line number information from an object file
stty	**stty**(1)	set the options for a terminal
stty	**stty**(1B)	set the options for a terminal
sum	**sum**(1)	print checksum and block count for a file
sum	**sum**(1B)	calculate a checksum for a file
sun	**machid**(1)	get processor type truth value
suspend	**suspend**(1)	shell built-in function to halt the current shell
switch	**case**(1)	shell built-in functions to choose from among a list of actions
symorder	**symorder**(1)	rearrange a list of symbols
sysV-make	**sysV-make**(1)	maintain, update, and regenerate groups of programs
t300	**plot**(1B)	graphics filters for various plotters
t300s	**plot**(1B)	graphics filters for various plotters
t4013	**plot**(1B)	graphics filters for various plotters
t450	**plot**(1B)	graphics filters for various plotters
tabs	**tabs**(1)	set tabs on a terminal
tail	**tail**(1)	deliver the last part of a file
talk	**talk**(1)	talk to another user
tar	**tar**(1)	create tape archives, and add or extract files
tbl	**tbl**(1)	format tables for nroff or troff
tcopy	**tcopy**(1)	copy a magnetic tape
tee	**tee**(1)	replicate the standard output
tek	**plot**(1B)	graphics filters for various plotters
telnet	**telnet**(1)	user interface to a remote system using the TELNET protocol
test	**if**(1)	shell built-in functions to evaluate expression(s) or to make execution of actions dependent upon the evaluation of expression(s)
test	**test**(1B)	condition evaluation command
test	**test**(1F)	condition evaluation command
tftp	**tftp**(1)	trivial file transfer program
time	**time**(1)	time a command

times	**times**(1)	shell built-in function to report time usages of the current shell
timex	**timex**(1)	time a command; report process data and system activity
tip	**tip**(1)	connect to remote system
touch	**touch**(1)	update access time and/or modification time of a file
tput	**tput**(1)	initialize a terminal or query terminfo database
tr	**tr**(1)	translate characters
tr	**tr**(1B)	translate characters
trap	**trap**(1)	shell built-in functions to respond to (hardware) signals
troff	**troff**(1)	typeset or format documents
true	**true**(1)	provide truth values
truss	**truss**(1)	trace system calls and signals
tset	**tset**(1B)	establish or restore terminal characteristics
tsort	**tsort**(1)	topological sort
tty	**tty**(1)	get the name of the terminal
type	**typeset**(1)	shell built-in functions to set/get attributes and values for shell variables and functions
typeset	**typeset**(1)	shell built-in functions to set/get attributes and values for shell variables and functions
u370	**machid**(1)	get processor type truth value
u3b15	**machid**(1)	get processor type truth value
u3b2	**machid**(1)	get processor type truth value
u3b5	**machid**(1)	get processor type truth value
u3b	**machid**(1)	get processor type truth value
ucblinks	**ucblinks**(1B)	adds /dev entries to give SunOS 4.1 compatible names to SunOS 5.x devices
ul	**ul**(1)	do underlining
ulimit	**limit**(1)	shell built-in functions to set/get limitations on the system resources available to the current shell and it’s descendents.
umask	**umask**(1)	shell built-in function to restrict read/write/execute permissions
unalias	**alias**(1)	shell built-in functions to create your own pseudonym or shorthand for a command or series of commands
uname	**uname**(1)	print name of current system
uncompress	**compress**(1)	compress, uncompress files or display expanded files
unexpand	**expand**(1)	expand TAB characters to SPACE characters, and vice versa
unget	**sccs-unget**(1)	undo a previous get of an SCCS file

unhash	**hash**(1)	shell built-in functions to evaluate the internal hash table of the contents of directories
unifdef	**unifdef**(1)	resolve and remove ifdef'ed lines from C program source
uniq	**uniq**(1)	report repeated lines in a file
units	**units**(1)	converts quantities expressed in standard scales to other scales
unix2dos	**unix2dos**(1)	convert text file from ISO format to DOS format
unlimit	**limit**(1)	shell built-in functions to set / get limitations on the system resources available to the current shell and it's descendents.
unpack	**pack**(1)	compress and expand files
unset	**set**(1)	shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents
unset	**set**(1F)	set and unset local or global environment variables
unsetenv	**set**(1)	shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents
until	**while**(1)	shell built-in functions to repetitively execute a set of actions while / until conditions are evaluated TRUE
uptime	**uptime**(1)	show how long the system has been up
users	**users**(1B)	display a compact list of users logged in
uucp	**uucp**(1C)	UNIX-to-UNIX system copy
uudecode	**uuencode**(1C)	encode a binary file, or decode its ASCII representation
uuencode	**uuencode**(1C)	encode a binary file, or decode its ASCII representation
uuglist	**uuglist**(1C)	print the list of service grades that are available on this UNIX system
uulog	**uucp**(1C)	UNIX-to-UNIX system copy
uuname	**uucp**(1C)	UNIX-to-UNIX system copy
uupick	**uuto**(1C)	public UNIX-to-UNIX system file copy
uustat	**uustat**(1C)	uucp status inquiry and job control
uuto	**uuto**(1C)	public UNIX-to-UNIX system file copy
uux	**uux**(1C)	UNIX-to-UNIX system command execution
vacation	**vacation**(1)	reply to mail automatically
val	**sccs-val**(1)	validate an SCCS file
valdate	**ckdate**(1)	prompts for and validates a date
valgid	**ckgid**(1)	prompts for and validates a group id

valint	**ckint**(1)	display a prompt; verify and return an integer value
valpath	**ckpath**(1)	display a prompt; verify and return a pathname
valrange	**ckrange**(1)	prompts for and validates an integer
valstr	**ckstr**(1)	display a prompt; verify and return a string answer
valtime	**cktime**(1)	display a prompt; verify and return a time of day
valuid	**ckuid**(1)	prompts for and validates a user ID
valyorn	**ckyorn**(1)	prompts for and validates yes / no
vax	**machid**(1)	get processor type truth value
vc	**vc**(1)	version control
vedit	**vi**(1)	screen-oriented (visual) display editor based on ex
vgrind	**vgrind**(1)	grind nice program listings
vi	**vi**(1)	screen-oriented (visual) display editor based on ex
view	**vi**(1)	screen-oriented (visual) display editor based on ex
vipw	**vipw**(1B)	edit the password file
volcancel	**volcancel**(1)	cancel user’s request for removable media that is not currently in drive
volcheck	**volcheck**(1)	check for media in a drive. Default checks all floppy media.
volmissing	**volmissing**(1)	notify user that volume requested is not in the CD-ROM or floppy drive
vplot	**plot**(1B)	graphics filters for various plotters
vsig	**vsig**(1F)	synchronize a co-process with the controlling FMLI application
w	**w**(1)	who is logged in, and what are they doing
wait	**wait**(1)	shell built-in function to wait for other jobs or processes
wc	**wc**(1)	display a count of lines, words and characters in a file
what	**what**(1)	extract SCCS version information from a file
whatis	**whatis**(1)	display a one-line summary about a keyword
whence	**typeset**(1)	shell built-in functions to set / get attributes and values for shell variables and functions
whereis	**whereis**(1B)	locate the binary, source, and manual page files for a command
which	**which**(1)	locate a command; display its pathname or alias

while	**while**(1)	shell built-in functions to repetitively execute a set of actions while / until conditions are evaluated TRUE
who	**who**(1)	who is on the system
whoami	**whoami**(1B)	display the effective current username
whois	**whois**(1)	Internet user name directory service
write	**write**(1)	write to another user
xargs	**xargs**(1)	construct argument list(s) and execute command
xgettext	**xgettext**(1)	extract gettext call strings from C programs
xstr	**xstr**(1)	extract strings from C programs to implement shared strings
yacc	**yacc**(1)	yet another compiler-compiler
ypcat	**ypcat**(1)	print values in a NIS database
ypmatch	**ypmatch**(1)	print the value of one or more keys from a NIS map
yppasswd	**yppasswd**(1)	change your network password in the NIS database
ypwhich	**ypwhich**(1)	return name of NIS server or map master
zcat	**compress**(1)	compress, uncompress files or display expanded files

NAME

acctcom – search and print process accounting files

SYNOPSIS

acctcom [–**abfhikmqrtv**] [–**C** *sec*] [–**e** *time*] [–**E** *time*] [–**g** *group*] [–**H** *factor*]
[–**I** *chars*] [–**l** *line*] [–**n** *pattern*] [–**o** *output-file*] [–**O** *sec*] [–**s** *time*]
[–**S** *time*] [–**u** *user*] [*filename* . . .]

AVAILABILITY

SUNWaccu

DESCRIPTION

acctcom reads *filename*s, the standard input, or **/var/adm/pacct**, in the form described by **acct**(4) and writes selected records to standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME**, **USER**, **TTYNAME**, **START TIME**, **END TIME**, **REAL (SEC)**, **CPU (SEC)**, **MEAN SIZE (K)**, and optionally, **F** (the **fork( )** / **exec( )** flag: **1** for **fork( )** without **exec( )**), **STAT** (the system exit status), **HOG FACTOR**, **KCORE MIN**, **CPU FACTOR**, **CHARS TRNSFD**, and **BLOCKS READ** (total blocks read and written).

A ' **#** ' is prepended to the command name if the command was executed with super-user privileges. If a process is not associated with a known terminal, a ' **?** ' is printed in the **TTYNAME** field.

If no *filename* is specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using ' **&** ' in the shell), **/var/adm/pacct** is read; otherwise, the standard input is read.

If any *filename* arguments are given, they are read in their respective order. Each file is normally read forward, that is, in chronological order by process completion time. The file **/var/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/var/adm/pacct***incr*.

OPTIONS

–**a**	Show some average statistics about the processes selected. The statistics will be printed after the output records.
–**b**	Read backwards, showing latest commands first. This option has no effect when standard input is read.
–**f**	Print the **fork()** / **exec()** flag and system exit status columns in the output. The numeric output for this option will be in octal.
–**h**	Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as (total CPU time) / (elapsed time).
–**i**	Print columns containing the I / O counts in the output.
–**k**	Instead of memory size, show total kcore-minutes.
–**m**	Show mean core size (the default).
–**q**	Do not print any output records, just print the average statistics as with the –**a** option.
–**r**	Show CPU factor (user-time / (system-time + user-time)).

–**t**	Show separate system and user CPU times.
–**v**	Exclude column headings from the output.
–**C** *sec*	Show only processes with total CPU time (system-time + user-time) exceeding *sec* seconds.
–**e** *time*	Select processes existing at or before *time*.
–**E** *time*	Select processes ending at or before *time*. Using the same *time* for both –**S** and –**E** shows the processes that existed at *time*.
–**g** *group*	Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.
–**H** *factor*	Show only processes that exceed *factor*, where factor is the “hog factor” as explained in option –**h** above.
–**I** *chars*	Show only processes transferring more characters than the cutoff number given by *chars*.
–**l** *line*	Show only processes belonging to terminal **/dev/term/***line*.
–**n** *pattern*	Show only commands matching *pattern* that may be a regular expression as in **regcmp**(3G), except + means one or more occurrences.
–**o** *output-file*	Copy selected process records in the input data format to *output-file*; suppress printing to standard output.
–**O** *sec*	Show only processes with CPU system time exceeding *sec* seconds.
–**s** *time*	Select processes existing at or after *time*, given in the format *hr* [:*min* [:*sec*]].
–**S** *time*	Select processes starting at or after *time*.
–**u** *user*	Show only processes belonging to *user*. The user may be specified by a user ID, a login name that is then converted to a user ID, ‘ **#** ’ (which designates only those processes executed with superuser privileges), or ‘ **?** ’ (which designates only those processes associated with unknown user IDs).

FILES

/etc/group
/etc/passwd
/var/adm/pacct*incr*

SEE ALSO

ps(1), **acct**(1M), **acctcms**(1M), **acctcon**(1M), **acctmerg**(1M), **acctprc**(1M), **acctsh**(1M), **fwtmp**(1M), **runacct**(1M), **su**(1M), **acct**(2), **regcmp**(3G), **acct**(4), **utmp**(4)

Security, Performance, and Accounting Administration

NOTES

acctcom reports only on processes that have terminated; use **ps**(1) for active processes.

If *time* exceeds the present time, then *time* is interpreted as occurring on the previous day.

NAME

adb – general-purpose debugger

SYNOPSIS

adb [**–w**] [**–k**] [**–I** *dir*] [**–P** *prompt*] [*objectfile* [*corefile* [*swapfile*]]]

AVAILABILITY

SUNWtoo

DESCRIPTION

adb is an interactive, general-purpose debugger. It can be used to examine files and provides a controlled environment for the execution of programs.

objectfile is normally an executable program file, preferably containing a symbol table. If the file does not contain a symbol table, it can still be examined, but the symbolic features of **adb** cannot be used. The default for *objectfile* is **a.out**. *corefile* is assumed to be a core image file produced after executing *objectfile*. The default for *corefile* is **core**. *swapfile* is the image of the swap device used. It is valid only when used with the **–k** option.

OPTIONS

–w	Create both *objectfile* and *corefile*, if necessary, and open them for reading and writing so that they can be modified using **adb**.
–k	Perform kernel memory mapping; use when *corefile* is a system crash dump or **/dev/mem**, or when using a *swapfile*.
–I *dir*	Specify a directory where files to be read with **$<** or **$<<** (see below) will be sought; the default is **/usr/kvm/lib/adb**.
–P *prompt*	Specify the **adb** prompt string.

USAGE

adb reads commands from the standard input and displays responses on the standard output. It does not supply a prompt by default. It ignores the QUIT signal. INTERRUPT invokes the next **adb** command. **adb** generally recognizes command input of the form:

[*address*] [**,** *count*] [*command*] [**;**]

address and *count* (if supplied) are expressions that result, respectively, in a new current address, and a repetition count. *command* is composed of a verb followed by a modifier or list of modifiers.

The symbol ‘**.**’ represents the current location. It is initially zero. The default *count* is ‘**1**’.

Expressions

.	The value of *dot*.
+	The value of *dot* incremented by the current increment.
^	The value of *dot* decremented by the current increment.
&	The last *address* typed. (In older versions of **adb**, ‘"’ was used.)
integer	A number. The prefixes **0o** and **0O** indicate octal; **0t** and **0T**, decimal; **0x** and **0X**, hexadecimal (the default).
*int***.***frac*	A floating-point number.
’*cccc***’**	ASCII value of up to 4 characters.
<*name*	The value of *name*, which is either a variable name or a register name.
symbol	A symbol in the symbol table.
(*exp***)**	The value of *exp*.

Unary Operators

**exp*	The contents of location *exp* in *corefile*.
%*exp*	The contents of location *exp* in *objectfile* (In older versions of **adb**, ‘**@**’ was used).
–exp	Integer negation.
~*exp*	Bitwise complement.
#*exp*	Logical negation.

Binary Operators

Binary operators are left associative and have lower precedence than unary operators.

+	Integer addition.
–	Integer subtraction.
*	Integer multiplication.
%	Integer division.
&	Bitwise conjunction (“AND”).
|	Bitwise disjunction (“OR”).
#	*lhs* rounded up to the next multiple of *rhs*.

Variables

Named variables are set initially by **adb** but are not used subsequently.

0	The last value printed.
1	The last offset part of an instruction source.
2	The previous value of variable 1.
9	The count on the last **$<** or **$<<** command.

On entry the following are set from the system header in the *corefile* or *objectfile* as appropriate.

b	The base address of the data segment.
d	The data segment size.
e	The entry point.
m	The ‘magic’ number
t	The text segment size.

Commands

Commands to **adb** consist of a *verb* followed by a *modifier* or list of modifiers.

Verbs

?	Print locations starting at *address* in *objectfile*.
/	Print locations starting at *address* in *corefile*.
=	Print the value of *address* itself.
:	Manage a subprocess.
>	Assign a value to a variable or register.
RETURN	Repeat the previous command with a *count* of 1. Increment ‘**.**’.
!	Shell escape.

?, /, *and* = *Modifiers*

The following format modifiers apply to the commands **?**, **/**, and **=**. To specify a format, follow the command with an optional repeat count, and the desired format letter or letters:

{ **?,/,=** } [[*r*] *f* . . .]]

where *r* is a repeat count, and *f* is one of the format letters listed below:

o	('**.**' increment: 2) Print 2 bytes in octal.
O	(4) Print 4 bytes in octal.
q	(2) Print in signed octal.
Q	(4) Print long signed octal.
d	(2) Print in decimal.
D	(4) Print long decimal.
x	(2) Print 2 bytes in hexadecimal.
X	(4) Print 4 bytes in hexadecimal.
u	(2) Print as an unsigned decimal number.
U	(4) Print long unsigned decimal.
f	(4) Print a single-precision floating-point number.
F	(8) Print a double-precision floating-point number.
b	(1) Print the addressed byte in octal.
c	(1) Print the addressed character.
C	(1) Print the addressed character using ˆ escape convention.
s	(*n*) Print the addressed string.
S	(*n*) Print a string using the ˆ escape convention.
Y	(4) Print 4 bytes in date format.
i	(*4* on SPARC; *n* on x86) Print as machine instructions.
a	(0) Print the value of '**.**' in symbolic form.
p	(4) Print the addressed value in symbolic form.
t	(0) Tab to the next appropriate TAB stop.
r	(0) Print a SPACE.
n	(0) Print a NEWLINE.
"..."	(0) Print the enclosed string.
ˆ	(0) Decrement '**.**'.
+	(0) Increment '**.**'.
–	(0) Decrement '**.**' by 1.

? *and* **/** *Modifiers*

l *value mask*	Apply *mask* and compare for *value*; move '**.**' to matching location.
L *value mask*	Apply *mask* and compare for 4-byte *value*; move '**.**' to matching location.
w *value*	Write the 2-byte *value* to address.
W *value*	Write the 4-byte *value* to address.
m *b1 e1 f1*[?]	Map new values for *b1, e1, f1*. If the **?** or **/** is followed by ∗ then the second segment (*b2* , *e2* , *f2*) of the address mapping is changed.
v	Like **w**, but writes only bytes at a time.

: *Modifiers*	**b** *commands*	Set breakpoint, execute *commands* when reached.
	r	Run *objectfile* as a subprocess.
	d	Delete breakpoint at *address*.
	z	Delete all breakpoints.
	c*s*	x86: The subprocess is continued with signal *s*.
	s*s*	Single-step the subprocess with signal *s*.
	i	Add the signal specified by *address* to the list of signals passed directly to the subprocess.
	t	Remove the signal specified by *address* from the list implicitly passed to the subprocess.
	k	Terminate (kill) the current subprocess, if any.
	A	Attach **adb** to an existing process id. (For example, **0t1234:A** would attach **adb** to decimal process number **1234**.)
	R	Release the previously attached process.
$ *Modifiers*	<*filename*	Read commands from the file *filename*.
	<<*filename*	Similar to **<**, but can be used in a file of commands without closing the file.
	>*filename*	Append output to *filename*, which is created if it does not exist.
	l	x86: Show the current lightweight process (lwp) ID.
	L	x86: Show all the lwp IDs.
	P	Specify the **adb** prompt string.
	?	Print process ID, the signal which stopped the subprocess, and the registers.
	r	Print the names and contents of the general CPU registers, and the instruction addressed by **pc**.
	x or **X**	x86: Print the contents of floating point registers. **$x** and **$X** accept a "count" which determines the precision in which the floating point registers will be printed; the default is **25**. Using **$X** will produce more verbose output than using **$x**.
	x	SPARC: Print the names and contents of floating-point registers 0 through 15.
	X	SPARC: Print the names and contents of floating-point registers 16 through 31.
	b	Print all breakpoints and their associated counts and commands.
	c	C stack backtrace. On Sun-4 systems, it is impossible for **adb** to determine how many parameters were passed to a function. The default that **adb** chooses in a **$c** command is to show the six parameter registers. This can be overridden by appending a hexadecimal number to the **$c** command, specifying how many parameters to display. For example, the **$cf** command will print 15 parameters for each function in the stack trace.
	C	x86: Same as **$c**, but in addition it displays the frame pointer values.
	d	Set the default radix to *address* and report the new value. Note: *address* is interpreted in the (old) current radix. Thus '**10$d**' never changes the default radix.

e	Print the names and values of external variables.
w	Set the page width for output to *address* (default 80).
s	Set the limit for symbol matches to *address* (default 255).
o	All integers input are regarded as octal.
q	Exit from **adb**.
v	Print all non zero variables in octal.
m	Print the address map.
f	Print a list of known source filenames.
p	(*Kernel debugging*) Change the current kernel memory mapping to map the designated **user structure** to the address given by *u ;* this is the address of the user's **proc** structure.
i	Show which signals are passed to the subprocess with the minimum of **adb** interference.
W	Reopen *objectfile* and *corefile* for writing, as though the −**w** command-line argument had been given.

EXAMPLES

To start **adb** on the running kernel, use (as **root**):

```
example# adb -k /dev/ksyms /dev/mem
```

/dev/ksyms is a special driver that provides an image of the kernel's symbol table. This can be used to examine kernel state and debug device drivers. Refer to the Debugging chapter in *Writing Device Drivers* for more information.

FILES

/usr/kvm/lib/adb	default directory in which files are to be read with **$<** and **$<<**.
a.out	
core	
/dev/ksyms	special driver to provide an image of the kernel's symbolic table.

SEE ALSO

ptrace(2), **a.out**(4), **core**(4), **proc**(4), **ksyms**(7)

Writing Device Drivers

DIAGNOSTICS

adb, when there is no current command or format, comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is **0**, unless last command failed or returned nonzero status.

NOTES

adb should be changed to use the new format symbolic information generated by −**g**.

adb is platform and release dependent. Kernel core dumps should be examined on the same platform they were created on.

BUGS

On SPARC systems, there does not seem to be any way to clear all breakpoints.

Since no shell is invoked to interpret the arguments of the **:r** command, the customary wild-card and variable expansions cannot occur.

Since there is little type-checking on addresses, using a sourcefile address in an inappropriate context may lead to unexpected results.

The **$c***parameter-count* command is a workaround.

NAME

addbib – create or extend a bibliographic database

SYNOPSIS

addbib [–**a**] [–**p** *promptfile*] *database*

AVAILABILITY

SUNWdoc

DESCRIPTION

When **addbib** starts up, answering **y** to the initial **Instructions?** prompt yields directions; typing **n** or RETURN skips them. **addbib** then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to *database.* A null response (just RETURN) means to leave out that field. A '–' (minus sign) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating **Continue?** prompt allows the user either to resume by typing **y** or RETURN, to quit the current session by typing **n** or **q**, or to edit *database* with any system editor (see **vi**(1), **ex**(1), **ed**(1)).

OPTIONS

–**a**	Suppress prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL–D.
–**p** *promptfile*	Use a new prompting skeleton, defined in *promptfile*. This file should contain prompt strings, a TAB, and the key-letters to be written to the *database*.

USAGE

Bibliography Key Letters

The most common key-letters and their meanings are given below. **addbib** insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

%A	Author's name
%B	Book containing article referenced
%C	City (place of publication)
%D	Date of publication
%E	Editor of book containing article referenced
%F	Footnote number or label (supplied by **refer**)
%G	Government order number
%H	Header commentary, printed before reference
%I	Issuer (publisher)
%J	Journal containing article
%K	Keywords to use in locating reference
%L	Label field used by –**k** option of **refer**
%M	Bell Labs Memorandum (undefined)
%N	Number within volume
%O	Other commentary, printed at end of reference

%P	Page number(s)
%Q	Corporate or Foreign Author (unreversed)
%R	Report, paper, or thesis (unpublished)
%S	Series title
%T	Title of article or book
%V	Volume number
%X	Abstract — used by **roffbib**, not by **refer**
%Y,Z	Ignored by **refer**

EXAMPLES

Except for A**,** each field should be given just once. Only relevant fields should be supplied.

```
%A	Mark Twain
%T	Life on the Mississippi
%I	Penguin Books
%C	New York
%D	1978
```

SEE ALSO

ed(1), **ex**(1), **indxbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1), **vi**(1)

NAME

alias, unalias – shell built-in functions to create your own pseudonym or shorthand for a command or series of commands

SYNOPSIS

csh

alias [*name* [*def*]]

unalias *pattern*

ksh

†† **alias** [–**tx**] [*name*[**=***value*]] . . .

unalias *name***. . .**

DESCRIPTION

csh

alias assigns *def* to the alias *name*. *def* is a list of words that may contain escaped history-substitution metasyntax. *name* is not allowed to be **alias** or **unalias**. If *def* is omitted, the alias *name* is displayed along with its current definition. If both *name* and *def* are omitted, all aliases are displayed.

unalias discards aliases that match (filename substitution) *pattern*. All aliases may be removed by '**unalias** *'.

ksh

alias with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The –**t** flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the –**t** flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The –**x** flag is used to set or print *exported alias*es. An *exported alias* is defined for scripts invoked by name. The exit status is non-zero if a *name* is given, but no value, and no alias has been defined for the *name*.

The *alias*es given by the list of *name*s may be removed from the *alias* list with **unalias**.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

csh(1), **ksh**(1), **shell_builtins**(1),

NAME

apropos – locate commands by keyword lookup

SYNOPSIS

apropos *keyword*...

AVAILABILITY

SUNWdoc

DESCRIPTION

apropos displays the man page name, section number, and a short description for each man page whose NAME line contains *keyword*. This information is contained in the **/usr/share/man/windex** database created by **catman**(1M). If **catman**(1M) was not run, or was run with the –**n** option, **apropos** fails. Each word is considered separately and the case of letters is ignored. Words which are part of other words are considered; for example, when looking for ‘compile’, **apropos** finds all instances of ‘compiler’ also.

apropos is actually just the –**k** option to the **man**(1) command.

Try

```
example% apropos password
```

and

```
example% apropos editor
```

If the line starts ‘*filename*(*section*) ...’ you can do ‘**man –s** *section filename*’ to display the man page for *filename*. Try

```
example% apropos format
```

and then

```
example% man –s 3s printf
```

to get the manual page on the subroutine **printf( )**.

FILES

/usr/share/man/windex	table of contents and keyword database

SEE ALSO

man(1), **whatis**(1), **catman**(1M)

DIAGNOSTICS

/usr/share/man/windex: No such file or directory

This database does not exist. **catman**(1M) must be run to create it.

NAME

ar – maintain portable archive or library

SYNOPSIS

ar [–**V**] – *key* [*arg*] [*posname*] *afile* [*name*. . .]

DESCRIPTION

The **ar** command maintains groups of files combined into a single archive file. Its main use is to create and update library files. However, it can be used for any similar purpose. The magic string and the file headers used by **ar** consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When **ar** creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure are described in detail in **ar**(4). The archive symbol table (described in **ar**(4)) is used by the link editor **ld** to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by **ar** when there is at least one object file in the archive. The archive symbol table is in a specially named file that is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the **ar** command is used to create or update the contents of such an archive, the symbol table is rebuilt. The **s** option described below will force the symbol table to be rebuilt.

OPTIONS

–V **ar** prints its version number on standard error.

Unlike command options, the *key* is a required part of the **ar** command line. The *key* is formed with one of the following letters: **drqtpmx**. Arguments to the *key*, alternatively, are made with one or more of the following set: **vuaibcs**. *posname* is an archive member name used as a reference point in positioning other files in the archive. *afile* is the archive file. The *name*s are constituent files in the archive file. The meanings of the *key* characters are as follows:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are listed. If names are given, only those files are listed.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

The meanings of the other key arguments are as follows:

v Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, print the filename preceding each extraction.

c Suppress the message that is produced by default when *afile* is created.

s Force the regeneration of the archive symbol table even if **ar**(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the **strip**(1) command has been used on the archive.

SEE ALSO

ld(1), **lorder**(1), **strip**(1), **a.out**(4), **ar**(4)

NOTES

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

By convention, archives are suffixed with the characters **.a**.

NAME

arch – display the architecture of the current host

SYNOPSIS

/usr/ucb/arch
/usr/ucb/arch –k
/usr/ucb/arch *archname*

DESCRIPTION

arch displays the application architecture of the current host system.

Systems can be broadly classified by their *architectures*, which define what executables will run on which machines. A distinction can be made between *kernel architecture* and *application architecture* (or, commonly, just “architecture”). Machines that run different kernels due to underlying hardware differences may be able to run the same application programs.

OPTIONS

–k	Display the kernel architecture, such as **sun4, sun4c**, etc. This defines which specific SunOS kernel will run on the machine, and has implications only for programs that depend on the kernel explicitly (for example, **ps**(1)).
archname	Return “true” (exit status **0**) if *application* binaries for *archname* can run on the current host system, otherwise, return “false” (exit status **1**). This is the preferred method for installation scripts to determine the environment of the host machine; that is, which architecture of a multi-architecture release to install on this machine. *archname* must be a valid application architecture.

SEE ALSO

mach(1B), **uname**(1)

NAME

as − assembler

SPARC SYNOPSIS

as [−**b**] [−**K PIC**] [−**L**] [−**m**] [−**n**] [−**o** *outfile*] [−**P** [[−**D***name*] [−**D***name=def*]
[−**I***path*] [−**U***name*]] . . .] [−**q**] [−**Q***y* | *n*] [−**s**] [−**S**[*a* | *C*]] [−**T**] [−**V**]
[−**xF**] *filename . . .*

x86 SYNOPSIS

as [−**m**] [−**n**] [−**o** *outfile*] [−**P** [[−**D***name*] [−**D***name=def*] [−**I***path*]
[−**U***name*]] . . .] [−**Q***y* | *n*] [−**s**] [−**V**] *filename . . .*

AVAILABILITY

SUNWsprot

DESCRIPTION

The **as** command creates object files from assembly language source *files*.

OPTIONS

The following flags may be specified in any order:

−**D***name* −**D***name=def*	When the −**P** option is in effect, these options are passed to the **cpp**(1) preprocessor without interpretation by the **as** command; otherwise, they are ignored.
−**I***path*	When the −**P** option is in effect, this option is passed to the **cpp**(1) preprocessor without interpretation by the **as** command; otherwise, it is ignored.
−**m**	Run the **m4**(1) macro processor on the input to the assembler.
−**n**	Suppress all the warnings while assembling.
−**o** *outfile*	Put the output of the assembly in *outfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.
−**P**	Run **cpp**(1), the C preprocessor, on the files being assembled. The preprocessor is run separately on each input file, not on their concatenation. The preprocessor output is passed to the assembler.
−**Qy** | **n**	Produce the "assembler version" information in the comment section of the output object file if the *y* option is specified; if the *n* option is specified, the information is suppressed.
−**s**	Place all stabs in the **.stabs** section. By default, stabs ares placed in **stabs.excl** sections, which are stripped out by the static linker, **ld**(1), during final execution. When the −**s** option is used, stabs remain in the final executable because **.stab** sections are not stripped by the static linker.
−**U***name*	When the −**P** option is in effect, this option is passed to the **cpp**(1) preprocessor without interpretation by the as command; otherwise, it is ignored.
−**V**	Write the version number of the assembler being run on the standard error output.

SPARC Options

–b	Enable Sun SourceBrowser.
–K PIC	Generate position-independent code.
–L	Save all symbols, including temporary labels that are normally discarded to save space, in the ELF symbol table.
–q	Perform a quick assembly. When the **–q** option is used, the assembler's internal node list is not built and the assembler simply emits instructions as they are read. **Note:** This option disables many error checks. It is recommended that you do not use this option to assemble handwritten assembly language.
–S[*a* | *C*]	Produce a disassembly of the emitted code to the standard output. • Adding the character *a* to the option appends a comment line to each assembly code which indicates its relative address in its own section. • Adding the character *C* to the option prevents comment lines from appearing in the output.
–T	This is a migration option for 4.1 assembly files to be assembled on 5.0 systems. With this option, the symbol names in 4.1 assembly files will be interpreted as 5.0 symbol names.
–xF	Generates additional information for performance analysis of the executable using SPARCworks analyzer. If the input file does not contain any stabs (debugging directives), then the assembler will generate some default stabs which are needed by the SPARCworks analyzer. Also see the manual pages analyzer(1), dbx(1), and collector(1).

ENVIRONMENT

TMPDIR	**as** normally creates temporary files in the directory **/tmp**. You may specify another directory by setting the environment variable **TMPDIR** to your chosen directory. (If **TMPDIR** isn't a valid directory, then **as** will use **/tmp**).

FILES

By default, **as** creates its temporary files in **/tmp**.

SEE ALSO

cc(1B), **cpp**(1), **ld**(1), **m4**(1), **nm**(1), **strip**(1), **tmpnam**(3S), **a.out**(4)

NOTES

If the **–m** (invoke the **m4**(1) macro processor) option is used, keywords for **m4**(1) cannot be used as symbols (variables, functions, labels) in the input file since **m4**(1) cannot determine which keywords are assembler symbols and which keywords are real **m4**(1) macros.

Whenever possible, you should access the assembler through a compilation system interface program such as **cc**(1B).

All undefined symbols are treated as global.

NAME

at, batch – execute commands at a later time

SYNOPSIS

at [–**csm**] [–**f** *script*] [–**q***queue*] *time* [*date*] [**+** *increment*]
at –**l** [*job*. . .]
at –**r** *job*. . .

batch

AVAILABILITY

SUNWcsu

DESCRIPTION

at and **batch** read commands from standard input to be executed at a later time. **at** allows you to specify when the commands should be executed, while jobs queued with **batch** will execute when system load level permits.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. No mail will be sent to the user if the job does not produce any standard output and / or standard error output (and –**m** option is not specified). The shell environment variables (except **TERM** and **TERMCAP**), current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost. The **SHELL** environment variable determines which shell is used.

at and **batch** write the job number and scheduled time to standard error.

at Access Control

Users: Access to **at** or **batch** is allowed:

- if the user's name appears in **/etc/cron.d/at.allow**.
- if **/etc/cron.d/at.allow** does not exist and the user's name is not in **/etc/cron.d/at.deny**.

Users: Access to **at** or **batch** is denied:

- if **/etc/cron.d/at.allow** exists and the user's name is not in it.
- if **/etc/cron.d/at.allow** does not exist and user's name is in **/etc/cron.d/at.deny**.
- if neither file exists.

Note: The rules for allow and deny apply to **root** only if the allow / deny files exist.

These files can only be modified by the privileged user.

time, date, increment Specifications

If the **DATEMSK** environment variable is set, it points to a template file that **at** will use to determine the valid *time* and *date* values instead of the values described below. For more information about using **DATEMSK**, see sub-section '**at and DATEMSK**'.

time	*time* may be specified as follows, where *h* is hours and *m* is minutes: *h*, *hh*, *hhmm*, *h*:*m*, *h*:*mm*, *hh*:*m*, *hh*:*mm*. A 24-hour clock is assumed, unless **am** or **pm** is appended to *time*. If **zulu** is appended to *time*, it means Greenwich Mean Time (GMT). *time* can also take on the values: **noon**, **midnight**, and **now**. **at now** responds with the error message **too late**; use **now** with the *increment* argument, such as: **at now + 1 minute**.
date	An optional *date* may be specified as either a month name followed by a

day number (and possibly a year number preceded by a comma) or a day of the week. (Both the month name and the day of the week may be spelled out or abbreviated to three characters.) Two special ‘‘days’’, **today** and **tomorrow** are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

increment The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.) The modifier **next** may precede the *increment;* it means ‘‘+ 1.’’

Thus valid commands include:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at now next day
at 5 pm Friday
```

Removing at and batch jobs

at –r removes jobs previously scheduled by **at** or **batch**. The job number is the number returned to you previously by the **at** or **batch** command. You can also get job numbers by typing **at –l**. You can only remove your own jobs unless you are the privileged user.

at and DATEMSK

If the environment variable **DATEMSK** is set, **at** will use its value as the full path name of a template file containing format strings. The strings consist of field descriptors and text characters and are used to provide a richer set of allowable date formats in different languages by appropriate settings of the environment variable **LANG** or **LC_TIME** (see **environ**(5)). (See **getdate**(3C) for the allowable list of field descriptors; this list is a subset of the descriptors allowed by **calendar**(1) that are listed on the **date**(1) manual page.) The formats described above for the *time* and *date* arguments, the special names **noon**, **midnight**, **now**, **next**, **today**, **tomorrow**, and the *increment* argument are not recognized when **DATEMSK** is set.

OPTIONS

–c	C shell. **csh**(1) is used to execute *script*. By default, the **SHELL** environment variable determines which shell to use.
–s	Standard (Bourne) shell. **sh**(1) is used to execute the job.
–m	Sends mail to the user after the job has been completed, indicating that the job is finished, even if the job produces no output. Mail is sent only if the job has not already generated a mail message.
–f *script*	Reads commands to be executed from the named *script* file.

–q*queue*	Submit the job in queue *queue* rather than the default queue **a**. The valid *queue*s are **a** through **z**. **batch** submits jobs in queue **b**. Queue **c** is reserved for **cron**(1M) and jobs cannot be submitted to that queue.
–l [*job*]	Reports all jobs scheduled for the invoking user, or just the *job*s specified. This option excludes the use of all other options.
–r *job*	Removes specified *job*s previously scheduled using **at**. This option excludes the use of all other options.

EXAMPLES

The **at** and **batch** commands read from standard input the commands to be executed at a later time. **sh**(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
example% batch
example% sort filename > outfile
<control-D> (hold down 'control' and depress 'd')
```

This sequence, which shows redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
example% batch <<!
example% sort filename 2>&1 > outfile | mail loginid
!
```

To have a job reschedule itself, invoke **at** from within the shell procedure, by including code similar to the following within the shell file:

```
example% echo "sh shellfile" | at 1900 thursday next week
```

The following example shows the possible contents of a template file **AT.TEMPL** in **/var/tmp**.

```
%I %p, the %est of %B of the year %Y run the following job
%I %p, the %end of %B of the year %Y run the following job
%I %p, the %erd of %B of the year %Y run the following job
%I %p, the %eth of %B of the year %Y run the following job
%d/%m/%y
%H:%M:%S
%I:%M%p
```

The following are examples of valid invocations if the environment variable **DATEMSK** is set to **/var/tmp/AT.TEMPL**.

```
at 2 PM, the 3rd of July of the year 2000 run the following job
at 3/4/99
at 10:30:30
at 2:30PM
```

FILES

/etc/cron.d	main cron directory
/etc/cron.d/at.allow	list of allowed users
/etc/cron.d/at.deny	list of denied users

/etc/cron.d/queuedefs	scheduling information
/var/spool/cron/atjobs	spool area for **at**.

SEE ALSO

atq(1), **atrm**(1), **calendar**(1), **crontab**(1), **date**(1), **kill**(1), **mail**(1), **nice**(1), **ps**(1), **sh**(1), **sort**(1), **cron**(1M), **getdate**(3C), **environ**(5)

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

atq – display the jobs queued to run at specified times

SYNOPSIS

atq [**–c**] [**–n**] [*username*. . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

atq displays the **at** jobs queued up for the current user. **at**(1) is a utility that allows users to execute commands at a later date. If invoked by the privileged user, **atq** will display all jobs in the queue.

If no options are given, the jobs are displayed in chronological order of execution.

When a privileged user invokes **atq** without specifying *username*, the entire queue is displayed; when a *username* is specified, only those jobs belonging to the named user are displayed.

OPTIONS

–c	Display the queued jobs in the order they were created (that is, the time that the **at** command was given).
–n	Display only the total number of jobs currently in the queue.

FILES

/var/spool/cron/atjobs	spool area for at jobs.

SEE ALSO

at(1), **atrm**(1), **cron**(1M)

NAME

atrm – remove jobs spooled by at or batch

SYNOPSIS

atrm [**–afi**] [[*job #*] [*user*] . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

atrm removes delayed-execution jobs that were created with the **at**(1) command, but have not yet executed. The list of these jobs and associated job numbers can be displayed by using **atq**(1).

atrm removes each job-number you specify, and / or all jobs belonging to the user you specify, provided that you own the indicated jobs.

You can only remove jobs belonging to other users if you have super-user privileges.

OPTIONS

–a	All. Remove all unexecuted jobs that were created by the current user. If invoked by the privileged user, the entire queue will be flushed.
–f	Force. All information regarding the removal of the specified jobs is suppressed.
–i	Interactive. **atrm** asks if a job should be removed. If you respond with a **y**, the job will be removed.

FILES

/var/spool/cron/atjobs	spool area for at jobs

SEE ALSO

at(1), **atq**(1), **cron**(1M)

NAME

audioconvert – convert audio file formats

SYNOPSIS

audioconvert [–**pF**] [–**f** *outfmt*] [–**o** *outfile*] [[–**i** *infmt*] [*file* . . .]] . . .

DESCRIPTION

audioconvert converts audio data between a set of supported audio encodings and file formats. It can be used to compress and decompress audio data, to add audio file headers to raw audio data files, and to convert between standard data encodings, such as µ-law and linear PCM.

If no filenames are present, **audioconvert** reads the data from the standard input stream and writes an audio file to the standard output. Otherwise, input files are processed in order, concatenated, and written to the output file.

Input files are expected to contain audio file headers that identify the audio data format. If the audio data does not contain a recognizable header, the format must be specified with the –**i** option, using the *rate*, *encoding*, and *channels* keywords to identify the input data format.

The output file format is derived by updating the format of the first input file with the format options in the –**f** specification. If –**p** is not specified, all subsequent input files are converted to this resulting format and concatenated together. The output file will contain an audio file header, unless *format=raw* is specified in the output format options.

Input files may be converted in place by using the –**p** option. When –**p** is in effect, the format of each input file is modified according to the –**f** option to determine the output format. The existing files are then overwritten with the converted data.

The **file**(1) command decodes and prints the audio data format of Sun audio files.

OPTIONS

–**p**	*In Place*: The input files are individually converted to the format specified by the –**f** option and rewritten. If a target file is a symbolic link, the underlying file will be rewritten. The –**o** option may not be specified with –**p**.
–**F**	*Force*: This option forces **audioconvert** to ignore any file header for input files whose format is specified by the –**i** option. If –**F** is not specified, **audioconvert** ignores the –**i** option for input files that contain valid audio file headers.
–**f** *outfmt*	*Output Format*: This option is used to specify the file format and data encoding of the output file. Defaults for unspecified fields are derived from the input file format. Valid keywords and values are listed in the next section.
–**o** *outfile*	*Output File*: All input files are concatenated, converted to the output format, and written to the named output file. If –**o** and –**p** are not specified, the concatenated output is written to the standard output. The –**p** option may not be specified with –**o**.

–i *infmt* *Input Format*: This option is used to specify the data encoding of raw input files. Ordinarily, the input data format is derived from the audio file header. This option is required when converting audio data that is not preceded by a valid audio file header. If **–i** is specified for an input file that contains an audio file header, the input format string will be ignored, unless **–F** is present. The format specification syntax is the same as the **–f** output file format.

Multiple input formats may be specified. An input format describes all input files following that specification, until a new input format is specified.

file *File Specification*: The named audio files are concatenated, converted to the output format, and written out. If no filename is present, or if the special filename '–' is specified, audio data is read from the standard input.

–? *Help*: Print a command line usage message.

FORMAT SPECIFICATION

The syntax for the input and output format specification is:

keyword=*value*[,*keyword*=*value* . . .]

with no intervening whitespace. Unambiguous values may be used without the preceding *keyword*=.

rate The audio sampling rate is specified in samples per second. If a number is followed by the letter **k**, it is multiplied by 1000 (for example, 44.1k = 44100). Standard of the commonly used sample rates are: 8k, 16k, 32k, 44.1k, and 48k.

channels The number of interleaved channels is specified as an integer. The words **mono** and **stereo** may also be used to specify one and two channel data, respectively.

encoding This option specifies the digital audio data representation. Encodings determine precision implicitly (**ulaw** implies 8-bit precision) or explicitly as part of the name (for example, **linear16**). Valid encoding values are:

ulaw CCITT G.711 μ-law encoding. This is an 8-bit format primarily used for telephone quality speech.

alaw CCITT G.711 A-law encoding. This is an 8-bit format primarily used for telephone quality speech in Europe.

linear8, linear16, linear32
Linear Pulse Code Modulation (PCM) encoding. The name identifies the number of bits of precision. **linear16** is typically used for high quality audio data.

pcm Same as **linear16**.

g721 CCITT G.721 compression format. This encoding uses Adaptive Delta Pulse Code Modulation (ADPCM) with 4-bit precision. It is primarily used for compressing µ-law voice data (achieving a 2:1 compression ratio).

g723 CCITT G.723 compression format. This encoding uses Adaptive Delta Pulse Code Modulation (ADPCM) with 3-bit precision. It is primarily used for compressing µ-law voice data (achieving an 8:3 compression ratio). The audio quality is similar to G.721, but may result in lower quality when used for non-speech data.

The following encoding values are also accepted as shorthand to set the sample rate, channels, and encoding:

voice Equivalent to **encoding=ulaw,rate=8k,channels=mono**.

cd Equivalent to **encoding=linear16,rate=44.1k,channels=stereo**.

dat Equivalent to **encoding=linear16,rate=48k,channels=stereo**.

format This option specifies the audio file format. Valid formats are:

sun Sun compatible file format (the default).

raw Use this format when reading or writing raw audio data (with no audio header), or in conjunction with an **offset** to import a foreign audio file format.

offset (**–i** *only*) Specify a byte offset to locate the start of the audio data. This option may be used to import audio data that contains an unrecognized file header.

EXAMPLES

Record voice data and compress it before storing it to a file:

```
example% audiorecord | audioconvert –f g721 > mydata.au
```

Concatenate two Sun format audio files, regardless of their data format, and output an 8-bit µ-law, 16 kHz, mono file:

```
example% audioconvert –f ulaw,rate=16k,mono –o outfile.au infile1 infile2
```

Convert a directory containing raw voice data files, in place, to Sun format (adds a file header to each file):

```
example% audioconvert –p –i voice –f sun *.au
```

SEE ALSO

audioplay(1), **audiorecord**(1), **file**(1)

NOTES

The algorithm used for converting multi-channel data to mono is implemented by simply summing the channels together. If the input data is perfectly in phase (as would be the case if a mono file is converted to stereo and back to mono), the resulting data may contain some distortion.

NAME

audioplay – play audio files

SYNOPSIS

audioplay [–**iV**] [–**v** *vol*] [–**b** *bal*] [–**p speaker** | **headphone** | **line**] [–**d** *dev*] [*file* . . .]

AVAILABILITY

SUNWaudio

DESCRIPTION

audioplay copies the named audio files (or the standard input if no filenames are present) to the audio device. If no input file is specified and standard input is a tty, the port, volume, and balance settings specified on the command line will be applied and the program will exit.

The input files must contain a valid audio file header. The encoding information in this header is matched against the capabilities of the audio device and, if the data formats are incompatible, an error message is printed and the file is skipped. Compressed ADPCM (G.721) monaural audio data is automatically uncompressed before playing.

Minor deviations in sampling frequency (that is, less than 1%) are ordinarily ignored. This allows, for instance, data sampled at 8012 Hz to be played on an audio device that only supports 8000 Hz. If the –**V** option is present, such deviations are flagged with warning messages.

OPTIONS

–**i** *Immediate*: If the audio device is unavailable (that is, another process currently has write access), **audioplay** ordinarily waits until it can obtain access to the device. When the –**i** option is present, **audioplay** prints an error message and exits immediately if the device is busy.

–**V** *Verbose*: Print messages on the standard error when waiting for access to the audio device or when sample rate deviations are detected.

–**v** *vol* *Volume*: The output volume is set to the specified value before playing begins, and is reset to its previous level when **audioplay** exits. The *vol* argument is an integer value between 0 and 100, inclusive. If this argument is not specified, the output volume remains at the level most recently set by any process.

–**b** *bal* *Balance*: The output balance is set to the specified value before playing begins, and is reset to its previous level when **audioplay** exits. The *bal* argument is an integer value between -100 and 100, inclusive. A value of -100 indicates left balance, 0 middle, and 100 right. If this argument is not specified, the output balance remains at the level most recently set by any process.

–**p speaker** | **headphone** | **line**
Output Port: Select the built-in **speaker**, (the default), **headphone** jack, or **line** out as the destination of the audio output signal. If this argument is not specified, the output port will remain unchanged. *Not all audio adapters support all of the output ports. If the named port does not exist, an appropriate substitute will be used.*

–**d** *dev* *Device*: The *dev* argument specifies an alternate audio device to which output should be directed. If the –**d** option is not specified, the **AUDIODEV** environment variable is consulted (see below). Otherwise, **/dev/audio** is used as the default audio device.

file	*File Specification*: Audio files named on the command line are played sequentially. If no filenames are present, the standard input stream (if it is not a tty) is played (it, too, must contain an audio file header). The special filename '–' may be used to read the standard input stream instead of a file. If a relative path name is supplied, the **AUDIOPATH** environment variable is consulted (see below).
–**?**	*Help*: Print a command line usage message.

ENVIRONMENT

AUDIODEV

The full path name of the audio device to write to, if no –d argument is supplied. If the **AUDIODEV** variable is not set, **/dev/audio** is used.

AUDIOPATH

A colon-separated list of directories in which to search for audio files whose names are given by relative pathnames. The current directory ("**.**") may be specified explicitly in the search path. If the **AUDIOPATH** variable is not set, only the current directory will be searched.

SEE ALSO

audioconvert(1), **audiorecord**(1)

Sparc Only **audio**(7), **audioamd**(7), **dbri**(7)

x86 Only **sbpro**(7)

BUGS

audioplay currently supports a limited set of audio format conversions. If the audio file is not in a format supported by the audio device, it must first be converted. For example, to convert to voice format on the fly, use the command:

```
example% audioconvert –f voice myfile | audioplay
```

The format conversion will not always be able to keep up with the audio output. If this is the case, you should convert to a temporary file before playing the data.

NAME

audiorecord – record an audio file

SYNOPSIS

audiorecord [**–af**] [**–v** *vol*] [**–b** *bal*] [**–m** *monvol*] [**–p** **mic** | **line** | **internal-cd**]
[**–c** *channels*] [**–s** *rate*] [**–e** *encoding*] [**–t** *time*] [**–i** *info*] [**–d** *dev*] [*file*]

AVAILABILITY

SUNWaudio

DESCRIPTION

audiorecord copies audio data from the audio device to a named audio file (or the standard output if no filename is present). If no output file is specified and standard output is a tty, the volume, balance, monitor volume, port, and audio format settings specified on the command line will be applied and the program will exit.

By default, monaural audio data is recorded at 8 kHz and encoded in μ-law format. If the audio device supports additional configurations, the **–c**, **–s**, and **–e** options may be used to specify the data format. The output file is prefixed by an audio file header that identifies the format of the data encoded in the file.

Recording begins immediately and continues until a **SIGINT** signal (for example, CTRL-C) is received. If the **–t** option is specified, **audiorecord** stops when the specified quantity of data has been recorded.

If the audio device is unavailable (that is, another process currently has read access), **audiorecord** prints an error message and exits immediately.

OPTIONS

–a *Append*: Append the data on the end of the named audio file. The audio device must support the audio data format of the existing file.

–f *Force*: When the **–a** flag is specified, the sample rate of the audio device must match the sample rate at which the original file was recorded. If the **–f** flag is also specified, sample rate differences are ignored, with a warning message printed on the standard error.

–v *vol* *Volume*: The recording gain is set to the specified value before recording begins, and is reset to its previous level when **audiorecord** exits. The *vol* argument is an integer value between 0 and 100, inclusive. If this argument is not specified, the input volume will remain at the level most recently set by any process.

–b *bal* *Balance*: The recording balance is set to the specified value before recording begins, and is reset to its previous level when **audiorecord** exits. The *bal* argument is an integer value between -100 and 100, inclusive. A value of -100 indicates left balance, 0 middle, and 100 right. If this argument is not specified, the input balance will remain at the level most recently set by any process.

−**m** *monvol*	*Monitor Volume*: The input monitor volume is set to the specified value before recording begins, and is reset to its previous level when **audiorecord** exits. The *monval* argument is an integer value between 0 and 100, inclusive. A non-zero value allows a directly connected input source to be heard on the output speaker while recording is in-progress. If this argument is not specified, the monitor volume will remain at the level most recently set by any process.
−**p mic** | **line** | **internal-cd**	*Input Port*: Select the **mic**, **line**, or **internal-cd** input as the source of the audio output signal. If this argument is not specified, the input port will remain unchanged. *Some systems will not support all possible input ports. If the named port does not exist, this option is ignored.*
−**c** *channels*	*Channels*: Specify the number of audio channels (1 or 2). The value may be specified as an integer or as the string **mono** or **stereo**. The default value is **mono**.
−**s** *rate*	*Sample Rate*: Specify the sample rate, in samples per second. If a number is followed by the letter **k**, it is multiplied by 1000 (for example, 44.1k = 44100). The default sample rate is 8 kHz.
−**e** *encoding*	*Encoding*: Specify the audio data encoding. This value may be one of **ulaw**, **alaw**, or **linear**. The default encoding is **ulaw**.
−**t** *time*	*Time*: The *time* argument specifies the maximum length of time to record. Time can be specified as a floating-point value, indicating the number of seconds, or in the form: *hh:mm:ss.dd*, where the hour and minute specifications are optional.
−**i** *info*	*Information*: The ‘information’ field of the output file header is set to the string specified by the *info* argument. This option cannot be specified in conjunction with the −**a** argument.
−**d** *dev*	*Device*: The *dev* argument specifies an alternate audio device from which input should be taken. If the −**d** option is not specified, the **AUDIODEV** environment variable is consulted (see below). Otherwise, **/dev/audio** is used as the default audio device.
file	*File Specification*: The named audio file is rewritten (or appended). If no filename is present (and standard output is not a tty), or if the special filename ‘–’ is specified, output is directed to the the standard output.
−**?**	*Help*: Print a command line usage message.

ENVIRONMENT

AUDIODEV	The full path name of the audio device to record from, if no −d argument is supplied. If the **AUDIODEV** variable is not set, **/dev/audio** is used.

SEE ALSO

audioconvert(1), **audioplay**(1)

SPARC Only **audio**(7), **audioamd**(7), **dbri**(7)

x86 Only **sbpro**(7)

NAME

awk – pattern scanning and processing language

SYNOPSIS

awk [–**f** *program-file*] [–**F***c*] [**'***prog***'**] [*parameters*] [*filename*. . .]

AVAILABILITY

SUNWesu

DESCRIPTION

awk scans each input *filename* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (′) to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *filename* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the –**f** *program-file* option. Input files are read in order; if there are no files, the standard input is read. The file name **'–'** means the standard input.

OPTIONS

–**f** *program-file*	**awk** uses the set of patterns it reads from *program-file*.
–**F***c*	Use the character *c* as the field separator (FS) character. See the discussion of **FS** below.

USAGE

Input Lines

Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. Any *filename* of the form *var=value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. *Variables* assigned in this manner are not available inside a **BEGIN** rule, and are assigned after previously specified files have been read.

An input line is normally made up of fields separated by white spaces. (This default can be changed by using the **FS** built-in variable or the –**F***c* option.) The default is to ignore leading blanks and to separate fields by blanks and / or tab characters. However, if **FS** is assigned a value that does not include any of the white spaces, then leading blanks are not ignored. The fields are denoted **$1**, **$2**, **. . .**; **$0** refers to the entire line.

Pattern-action Statements

A pattern-action statement has the form:

pattern **{** *action* **}**

Either pattern or action may be omitted. If there is no action, the matching line is printed. If there is no pattern, the action is performed on every input line. Pattern-action statements are separated by newlines or semicolons.

Patterns are arbitrary Boolean combinations (**!**,| | , **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

expression relop expression
expression matchop regular_expression

where a *relop* is any of the six relational operators in C, and a *matchop* is either **~** (contains) or **!~** (does not contain). An *expression* is an arithmetic expression, a relational expression, the special expression

var **in** array

or a Boolean combination of these.

Regular expressions are as in **egrep**(1). In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between the occurrence of the first pattern to the occurrence of the second pattern.

The special patterns **BEGIN** and **END** may be used to capture control before the first input line has been read and after the last input line has been read respectively. These keywords do not combine with any other patterns.

Built-in Variables

Built-in variables include:

FILENAME	name of the current input file
FS	input field separator regular expression (default blank and tab)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default **%.6g**)
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RS	input record separator (default new-line)

An action is a sequence of statements. A statement may be one of the following:

if (*expression*) *statement* [**else** *statement*]
while (*expression*) *statement*
do *statement* **while** (*expression*)
for (*expression* ; *expression* ; *expression*) *statement*
for (*var* **in** *array*) *statement*
break
continue
{ [*statement*] ... }
expression # commonly variable = expression
print [*expression-list*] [>*expression*]
printf *format* [, *expression-list*] [>*expression*]
next # skip remaining patterns on this input line
exit [**expr**] # skip the rest of the input; exit status is expr

Statements are terminated by semicolons, newlines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators **+**, **–**, ∗, **/**, **%**, **ˆ** and concatenation (indicated by a blank). The operators **++ –– += –= ∗= /= %= ˆ= > >= < <= == != ?:** are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (""), with the usual C escapes recognized within.

The **print** statement prints its arguments on the standard output, or on a file if *>expression* is present, or on a pipe if ’|*cmd’* is present. The output resulted from the print statement is terminated by the output record separator with each argument separated by the current output field separator. The **printf** statement formats its expression list according to the format (see **printf**(3S)).

The mathematical functions: **exp**, **log**, **sqrt**, are built-in.

Other built-in functions include:

index(*s*, *t***)**	returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.
int(*s***)**	truncates *s* to an integer value. If *s* is not specified, $0 is used.
length(*s***)**	returns the length of its argument taken as a string, or of the whole line if there is no argument.
match(*s*, *re***)**	returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all.
split(*s*, *a*, *fs***)**	splits the string *s* into array elements *a*[*1*], *a*[*2*], ... *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator **FS** if *fs* is not given.
sprintf(*fmt*, *expr*, *expr*, . . . **)**	formats the expressions according to the **printf**(3S) format given by *fmt* and returns the resulting string.
substr(*s*, *m*, *n***)**	returns the *n*-character substring of *s* that begins at position *m*.

The input/output built-in function is:

getline	sets **$0** to the next input record from the current input file. **getline** returns 1 for successful input, 0 for end of file, and –1 for an error.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN  { FS = ",[ \t]*|[ \t]+" }
       { print $2, $1 }
```

Add up first column, print sum and average:

```
        { s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; −−i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Print a file, filling in page numbers starting at 5:

```
/Page/  { $2 = n++; }
        { print }
```

Assuming this program is in a file named **prog**, the following command line prints the file **input** numbering its pages starting at 5: **awk −f prog n=5 input**.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **awk** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S style) locale determines how **awk** behaves.

LC_CTYPE
determines how **awk** handles characters. When **LC_CTYPE** is set to a valid value, **awk** can display and handle text and filenames containing valid characters for that locale. **awk** can display and handle Extended Unix Code (EUC) characters where any character can be 1, 2, or 3 bytes wide. **awk** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S/English).

SEE ALSO

egrep(1), **grep**(1), **nawk**(1), **sed**(1), **printf**(3S), **environ**(5)

NOTES

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

NAME

banner – make posters

SYNOPSIS

banner *strings*

AVAILABILITY

SUNWesu

DESCRIPTION

banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

echo(1)

NAME

basename, dirname – deliver portions of pathnames

SYNOPSIS

basename *string* [*suffix*]
dirname *string*

AVAILABILITY

SUNWcsu

DESCRIPTION

basename deletes any prefix ending in **/** and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (` `) within shell procedures. The *suffix* is a pattern as defined on the **ed**(1) manual page.

dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, invoked with the argument **/home/sms/personal/mail** sets the environment variable **NAME** to the file named **mail** and the environment variable **MYMAILPATH** to the string **/home/sms/personal**:

```
example% NAME=`basename  $HOME/personal/mail`
example% MYMAILPATH=`dirname  $HOME/personal/mail`
```

This shell procedure, invoked with the argument **/usr/src/bin/cat.c**, compiles the named file and moves the output to **cat** in the current directory:

```
example% cc $1
example% mv a.out `basename $1 .c`
```

SEE ALSO

ed(1), **sh**(1)

NAME

basename – display portions of pathnames

SYNOPSIS

/usr/ucb/basename *string* [*suffix*]

AVAILABILITY

SUNWscpu

DESCRIPTION

basename deletes any prefix ending in '/' and the *suffix*, if present in *string*. It directs the result to the standard output, and is normally used inside substitution marks (\` \`) within shell procedures. The *suffix* is a pattern as defined on **ed**(1).

EXAMPLES

This shell procedure invoked with the argument **/usr/src/bin/cat.c** compiles the named file and moves the output to **cat** in the current directory:

```
example% cc $1
example% mv a.out `basename $1 .c`
```

SEE ALSO

ed(1), **sh**(1)

NAME

bc – arbitrary precision arithmetic language

SYNOPSIS

bc [–**c**] [–**l**] [*filename*. . .]

AVAILABILITY

SUNWesu

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. **bc** is actually a preprocessor for the desk calculator program **dc**, which it invokes automatically unless the –**c** option is present. In this case the **dc** input is sent to the standard output instead.

The syntax for **bc** programs is as follows: *L* means letter **a**–**z**, *E* means expression, *S* means statement.

USAGE

Comments

Are enclosed in **/*** and ***/**.

Names

Simple variables: *L*.
Array elements: *L* [*E*].
The words **ibase**, **obase**, and **scale**.

Other Operands

Arbitrarily long numbers with optional sign and decimal point.
(*E* **)**
sqrt (*E* **)**
length (*E* **)** Number of significant decimal digits.
scale (*E* **)** Number of digits right of decimal point.
L **(** *E* , ... , *E* **)**

Operators

+ – * / % ^ (**%**is remainder; **^**is power)
++ –– (prefix and postfix; apply to names)
== <= >= != < >
= =+ =– =* =/ =% =^

Statements

E
{ *S* ;. . . ; *S* **}**
if (*E* **)** *S*
while (*E* **)** *S*
for (*E* ; *E* ; *E* **)** *S*
null statement
break
quit

Function Definitions

define *L* **(** *L* **,**. . . *L* **) {**
auto *L* **,**. . . *L*
S **;**. . . *S*
return (*E* **)**
}

Functions in –l Math Library

s(*x***)**	sine
c(*x***)**	cosine
e(*x***)**	exponential
l(*x***)**	log
a(*x***)**	arctangent
j(*n***,***x)*	Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to **scale** influences the number of digits to be retained on arithmetic operations in the manner of **dc**. Assignments to **ibase** or **obase** set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. **auto** variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

OPTIONS

–c	Compile only. The output is sent to the standard output.
–l	Argument stands for the name of an arbitrary precision math library.

EXAMPLES

Defines a function to compute an approximate value of the exponential function:

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

Prints approximate values of the exponential function of the first ten integers:

```
for(i=1; i<=10; i++) e(i)
```

FILES

/usr/bin/dc	desk calculator proper
/usr/lib/lib.b	mathematical library

SEE ALSO

dc(1)

NOTES

The **bc** command does not recognize the logical operators **&&** and | | .

The **for** statement must have all three expressions (*E*'s).

The **quit** statement is interpreted when read, not when executed.

NAME

bdiff – big diff

SYNOPSIS

bdiff *filename1 filename2* [*n*] [–**s**]

AVAILABILITY

SUNWesu

DESCRIPTION

bdiff is used in a manner analogous to **diff** to find which lines in *filename1* and *filename2* must be changed to bring the files into agreement. Its purpose is to allow processing of files too large for **diff**. If *filename1 (filename2)* is –, the standard input is read.

bdiff ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes **diff** on corresponding segments. If both optional arguments are specified, they must appear in the order indicated above.

The output of **bdiff** is exactly that of **diff**, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note: Because of the segmenting of the files, **bdiff** does not necessarily find a smallest sufficient set of file differences.

OPTIONS

n	The number of line segments. The value of *n* is 3500 by default. If the optional third argument is given and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for **diff**, causing it to fail.
–**s**	Specifies that no diagnostics are to be printed by **bdiff** (silent option). Note: However, this does not suppress possible diagnostic messages from **diff**, which **bdiff** calls.

FILES

/tmp/bd*?????*

SEE ALSO

diff(1)

DIAGNOSTICS

Use **help** for explanations.

NAME

biff – give notice of incoming mail messages

SYNOPSIS

/usr/ucb/biff [**y** | **n**]

AVAILABILITY

SUNWscpu

DESCRIPTION

biff turns mail notification on or off for the terminal session. With no arguments, **biff** displays the current notification status for the terminal.

If notification is allowed, the terminal rings the bell and displays the header and the first few lines of each arriving mail message. **biff** operates asynchronously. For synchronized notices, use the **MAIL** variable of **sh**(1) or the **mail** variable of **csh**(1).

A '**biff y**' command can be included in your **˜/.login** or **˜/.profile** file for execution when you log in.

OPTIONS

y	Allow mail notification for the terminal.
n	Disable notification for the terminal.

FILES

˜/.login
˜/.profile

SEE ALSO

csh(1), **mail**(1), **sh**(1)

NAME

break, continue – shell built-in functions to escape from or advance within a controlling while, for, foreach, or until loop

SYNOPSIS

sh

break [*n*]

continue [*n*]

csh

break

continue

ksh

† **break** [*n*]

† **continue** [*n*]

DESCRIPTION

sh

break exits from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.

continue resumes the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*-th enclosing loop.

csh

break resumes execution after the **end** of the nearest enclosing **foreach** or **while** loop. The remaining commands on the current line are executed. This allows multilevel breaks to be written as a list of **break** commands, all on one line.

continue continues execution of the next iteration of the nearest enclosing **while** or **foreach** loop.

ksh

break exits from the enclosed **for**, **while**, **until**, or **select** loop, if any. If *n* is specified then **break** *n* levels.

continue resumes the next iteration of the enclosed **for**, **while**, **until**, or **select** loop. If *n* is specified then resume at the *n*-th enclosed loop.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

csh(1), **exit**(1), **for**(1), **foreach**(1), **ksh**(1), **select**(1), **sh**(1), **until**(1), **while**(1)

NAME

cal – display a calendar

SYNOPSIS

cal [[*month*] *year*]

AVAILABILITY

SUNWesu

DESCRIPTION

cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. The calendar produced is that for England and the United States.

OPTIONS

month	The *month* is a number between 1 and 12.
year	The *year* can be between 1 and 9999.

SEE ALSO

calendar(1)

NOTES

An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type:

```
cal 9 1752
```

The command **cal 83** refers to the year 83, not 1983.

The year is always considered to start in January even though this is historically naive.

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

AVAILABILITY

SUNWesu

DESCRIPTION

calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as **Aug. 24**, **august 24**, **8/24**, and so forth, are recognized, but not **24 August** or **24/8**. On weekends ‘‘tomorrow’’ extends through Monday. **calendar** can be invoked regularly by using the **crontab**(1) or **at**(1) commands.

When an argument is present, **calendar** does its job for every user who has a file **calendar** in his or her login directory and sends them any positive results by **mail**(1). Normally this is done daily by facilities in the UNIX operating system (see **cron**(1M)).

If the environment variable **DATEMSK** is set, **calendar** will use its value as the full path name of a template file containing format strings. The strings consist of field descriptors and text characters and are used to provide a richer set of allowable date formats in different languages by appropriate settings of the environment variable **LANG** or **LC_TIME** (see **environ**(5)). (See **date**(1) for the allowable list of field descriptors.)

EXAMPLES

The following example shows the possible contents of a template:

%B %eth of the year %Y

%B represents the full month name, **%e** the day of month and **%Y** the year (4 digits).

If **DATEMSK** is set to this template, the following **calendar** file would be valid:

March 7th of the year 1989 < Reminder>

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **calendar** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **calendar** behaves.

LC_CTYPE

Determines how **calendar** handles characters. When **LC_CTYPE** is set to a valid value, **calendar** can display and handle text and filenames containing valid characters for that locale. **calendar** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **calendar** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_TIME

Determines how **calendar** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/etc/passwd	
/tmp/cal*	
/usr/lib/calprog	program used to figure out today's and tomorrow's dates

SEE ALSO

at(1), **crontab**(1), **date**(1), **mail**(1), **cron**(1M), **environ**(5)

NOTES

Appropriate lines beginning with white space will not be printed.

Your calendar must be public information for you to get reminder service.

calendar's extended idea of ''tomorrow'' does not account for holidays.

NAME

case, switch, select – shell built-in functions to choose from among a list of actions

SYNOPSIS

sh

case *word* **in** [*pattern* [| *pattern*] . . . **)** *actions* **;;**] . . . **esac**

csh

```
switch (expression)
        case comparison1:
                . . .
                breaksw
        case comparison2:
                . . .
                breaksw
        . . .
        default:
endsw
```

ksh

case *word* **in** [[**(**]*pattern* [| *pattern*] . . . **)** *actions* **;;**] . . . **esac**
select *identifier* [**in** *word* . . .] **; do** *list* **; done**

DESCRIPTION

sh

A **case** command executes the *actions* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

csh

The c-shell uses the **switch** statement, in which each *comparison* is successively matched, against the specified *expression*, which is first command and filename expanded. The file metacharacters ∗, **?** and **[. . .]** may be used in the case comparison, which are variable expanded. If none of the comparisons match before a “default” comparison is found, execution begins after the default comparison. Each **case** statement and the **default** statement must appear at the beginning of a line. The command **breaksw** continues execution after the **endsw**. Otherwise control falls through subsequent **case** and **default** statements as with C. If no comparison matches and there is no default, execution continues after the **endsw**.

case *comparison***:** A compared-expression in a **switch** statement.

default: If none of the preceeding *comparisons* match *expression*, then this is the default case in a **switch** statement. The default should come after all **case** comparisons. Any remaining commands on the command line are first executed.

breaksw exits from a **switch**, resuming after the **endsw**.

ksh

A **case** command executes the *actions* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see File Name Generation in **ksh**(1)).

A **select** command prints to standard error (file descriptor 2), the set of *word*s, each preceded by a number. If **in** *word* . . . is omitted, then the positional parameters are used instead. The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed *word*s, then the value of the variable *identifier* is set to the *word* corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the variable *identifier* is set to **NULL**. The contents of the line read from standard input is saved in the shell variable **REPLY**. The *list* is executed for each selection until a **break** or *end-of-file* is encountered. If the **REPLY** variable is set to **NULL** by the execution of *list*, then the selection list is printed before displaying the **PS3** prompt for the next selection.

SEE ALSO

break(1), **csh**(1), **ksh**(1), **sh**(1)

NAME

cat – concatenate and display files

SYNOPSIS

cat [–**bnsuvet**] *filename*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

cat reads each *filename* in sequence and writes it on the standard output. Thus:

```
example% cat filename
```

prints **filename** on your terminal, and:

```
example% cat filename1 filename2 >filename3
```

concatenates **filename1** and **filename2**, and writes the results in **filename3**.

If no input file is given, or if the argument – is encountered, **cat** reads from the standard input file.

OPTIONS

–**b**	Number the lines, as –**n**, but omit the line numbers from blank lines.
–**n**	Precede each line output with its line number.
–**u**	The output is not buffered. (The default is buffered output.)
–**s**	**cat** is silent about non-existent files.
–**v**	Non-printing characters (with the exception of tabs, new-lines and form-feeds) are printed visibly. ASCII control characters (octal 000 – 037) are printed as **ˆ***n*, where *n* is the corresponding ASCII character in the range octal 100 – 137 (@, A, B, C, . . ., X, Y, Z, [, \,], ˆ, and _); the DEL character (octal 0177) is printed **ˆ?**. Other non-printable characters are printed as **M-***x*, where *x* is the ASCII character specified by the low-order seven bits.

When used with the –**v** option, the following options may be used:

–**t**	Tabs will be printed as **ˆI**’s and formfeeds to be printed as **ˆL**’s.
–**e**	A **$** character will be printed at the end of each line (prior to the new-line).

The –**t** and –**e** options are ignored if the –**v** option is not specified.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **cat** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **cat** behaves.

LC_CTYPE
Determines how **cat** handles characters. When **LC_CTYPE** is set to a valid value, **cat** can display and handle text and filenames containing valid characters for that locale. **cat** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **cat** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

cp(1), **pg**(1), **pr**(1), **environ**(5)

NOTES

Redirecting the output of **cat** onto one of the files being read will cause the loss of the data originally in the file being read. For example,

```
example% cat filename1 filename2 >filename1
```

causes the original data in **filename1** to be lost.

NAME

cc – C compiler

SYNOPSIS

/usr/ucb/cc [*options*]

AVAILABILITY

SUNWscpu

DESCRIPTION

/usr/ucb/cc is the interface to the BSD Compatibility Package C compiler. It is a script that looks for the link **/usr/ccs/bin/ucbcc** to the C compiler. **/usr/ccs/bin/ucbcc** is available only with the SPROcc package, whose default location is **/opt/SUNWspro**. **/usr/ucb/cc** is identical to **/usr/ccs/bin/ucbcc**, except that BSD headers are used and BSD libraries are linked *before* base libraries. The **/opt/SUNWspro/man/man1/acc.1** man page is available only with the SPROcc package.

OPTIONS

/usr/ucb/cc accepts the same options as **/usr/ccs/bin/ucbcc**, with the following exceptions:

–**I***dir*	Search *dir* for included files whose names do not begin with a slash (/) prior to searching the usual directories. The directories for multiple –**I** options are searched in the order specified. The preprocessor first searches for **#include** files in the directory containing *sourcefile*, and then in directories named with –**I** options (if any), then **/usr/ucbinclude**, and finally, in **/usr/include**.
–**L***dir*	Add *dir* to the list of directories searched for libraries by **/usr/ccs/bin/ucbcc**. This option is passed to **/usr/ccs/bin/ld** and **/usr/css/lib**. Directories specified with this option are searched before **/usr/ucblib** and **/usr/lib**.
–**Y P**, *dir*	Change the default directory used for finding libraries.

FILES

/usr/ccs/bin/ld	link editor
/usr/lib/libc	C library
/usr/ucbinclude	BSD Compatibility directory for header files
/usr/ucblib	BSD Compatibility directory for libraries
/usr/ucblib/libucb	BSD Compatibility C library
/usr/lib/libsocket	library containing socket routines
/usr/lib/libnsl	library containing network functions
/usr/lib/libelf	library containing routines to process ELF object files
/usr/lib/libaio	library containing asynchronous I / O routines

SEE ALSO

ld(1), **a.out**(4)

NOTES

The –**Y P**, *dir* option may have unexpected results, and should not be used.

NAME

cd, chdir, pushd, popd, dirs – shell built-in functions to change the current working directory

SYNOPSIS

sh

cd [*argument*]

csh

cd [*dir*]
chdir [*dir*]
pushd [+*n* | *dir*]
popd [+*n*]
dirs [–**l**]

ksh

cd [*arg*]
cd *old new*

DESCRIPTION

sh

cd changes the current directory to *argument*. The shell parameter **HOME** is the default *argument*. The shell parameter **CDPATH** defines the search path for the directory containing *argument*. Alternative directory names are separated by a colon (**:**). The default path is **<null>** (specifying the current directory). Note: The current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *argument* begins with '**/**', '**.**', or '**..**', the search path is not used. Otherwise, each directory in the path is searched for *argument*. **cd** must have execute (search) permission in *argument*.
Because a new process is created to execute each command, **cd** would be ineffective if it were written as a normal command; therefore, it is recognized by and is internal to the shell. (See **pwd**(1), **sh**(1), and **chdir**(2)).

csh

If *dir* is not specified, **cd** uses the value of shell parameter **HOME** as the new working directory. If *dir* specifies a complete path starting with '**/**', '**.**', or '**..**', *dir* becomes the new working directory. If neither case applies, **cd** tries to find the designated directory relative to one of the paths specified by the **CDPATH** shell variable. **CDPATH** has the same syntax as, and similar semantics to, the **PATH** shell variable. **cd** must have execute (search) permission in *dir*. Because a new process is created to execute each command, **cd** would be ineffective if it were written as a normal command; therefore, it is recognized by and is internal to the C-shell. (See **pwd**(1), **sh**(1), and **chdir**(2)).

chdir changes the shell's working directory to directory *dir*. If no argument is given, change to the home directory of the user. If no argument is given, change to the home directory of the user. If *dir* is a relative pathname not found in the current directory, check for it in those directories listed in the **cdpath** variable. If *dir* is the name of a shell variable whose value starts with a **/**, change to the directory named by that value.

pushd will push a directory onto the directory stack. With no arguments, exchange the top two elements.

+*n* Rotate the *n*'th entry to the top of the stack and **cd** to it.

dir Push the current working directory onto the stack and change to *dir*.

popd pops the directory stack and **cd** to the new top directory. The elements of the directory stack are numbered from 0 starting at the top.

+*n* Discard the *n*'th entry in the stack.

dirs will print the directory stack, most recent to the left; the first directory shown is the current directory. With the **–l** argument, produce an unabbreviated printout; use of the **˜** notation is suppressed.

ksh The **cd** command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is – the directory is changed to the previous directory. The shell variable **HOME** is the default *arg*. The variable **PWD** is set to the current directory. The shell variable **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a NULL path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a ‘ **/** ’, ‘ **.** ’, or ‘ **..** ’, then the search path is not used. Otherwise, each directory in the path is searched for *arg*.

The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.

The **cd** command may not be executed by **rksh**. Because a new process is created to execute each command, **cd** would be ineffective if it were written as a normal command; therefore, it is recognized by and is internal to the Korn shell. (See **pwd**(1), **sh**(1), and **chdir**(2)).

SEE ALSO

csh(1), **ksh**(1), **pwd**(1), **sh**(1), **chdir**(2)

NAME

checknr – check nroff and troff input files; report possible errors

SYNOPSIS

checknr [**–fs**] [**–a .***x1* **.***y1* **.***x2* **.***y2* . . . **.***xn* **.***yn*] [**–c .***x1* **.***x2* **.***x3* . . . **.***xn*] [*filename* . . .]

AVAILABILITY

SUNWdoc

DESCRIPTION

checknr checks a list of **nroff**(1) or **troff**(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, **checknr** checks the standard input. Delimiters checked are:

- Font changes using **\f***x* . . . **\fP**.
- Size changes using **\s***x* . . . **\s0**.
- Macros that come in open . . . close forms, for example, the **.TS** and **.TE** macros which must always come in pairs.

checknr knows about the **ms**(5) and **me**(5) macro packages.

checknr is intended to be used on documents that are prepared with **checknr** in mind. It expects a certain document writing style for **\f** and **\s** commands, in that each **\f***x* must be terminated with **\fP** and each **\s***x* must be terminated with **\s0**. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from **checknr**. Since it is probably better to use the **\fP** and **\s0** forms anyway, you should think of this as a contribution to your document preparation style.

OPTIONS

–f Ignore **\f** font changes.

–s Ignore **\s** size changes.

–a .*x1* .*y1*. . .
Add pairs of macros to the list. The pairs of macros are assumed to be those (such as **.DS** and **.DE**) that should be checked for balance. The **–a** option must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair **.BS** and **.ES**, use '**–a.BS.ES**'

–c .*x1* . . .
Define commands which **checknr** would otherwise complain about as undefined.

SEE ALSO

eqn(1), **nroff**(1), **troff**(1), **me**(5), **ms**(5)

BUGS

There is no way to define a one-character macro name using the **–a** option.

NAME

chgrp – change the group ownership of a file

SYNOPSIS

chgrp [**–f**] [**–h**] [**–R**] *gid filename* . . .

AVAILABILITY

SUNWcsu

DESCRIPTION

chgrp changes the group ID (GID) of the *files* given as arguments to *group*. The group may be either a decimal group ID or a group name found in the group ID file, **/etc/group**.

You must be the owner of the file, or be the super-user to use this command. If you are not the super-user, and successfully change the group, the set-group-id bit is cleared.

The operating system has a configuration option { **_POSIX_CHOWN_RESTRICTED**}, to restrict ownership changes. When this option is in effect, the owner of the file may change the group of the file only to a group to which the owner belongs. Only the super-user can arbitrarily change owner **ID**s whether this option is in effect or not.

OPTIONS

–f	Force. Do not report errors.
–h	If the file is a symbolic link, change the group of the symbolic link. Without this option, the group of the file referenced by the symbolic link is changed.
–R	Recursive. **chgrp** descends through the directory, and any subdirectories, setting the specified group ID as it proceeds. When a symbolic link is encountered, the group of the target file is changed (unless the **–h** option is specified), but no recursion takes place.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **chgrp** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **chgrp** behaves.

LC_CTYPE

Determines how **chgrp** handles characters. When **LC_CTYPE** is set to a valid value, **chgrp** can display and handle text and filenames containing valid characters for that locale. **chgrp** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **chgrp** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/etc/group

SEE ALSO

chmod(1), **chown**(1), **id**(1M), **chown**(2), **group**(4), **passwd**(4), **environ**(5)

NAME

chkey – change user’s secure RPC key pair

SYNOPSIS

chkey [–**p**] [–**s nisplus** | **nis** | **files**]

AVAILABILITY

SUNWcsu

DESCRIPTION

chkey is used to change a user’s secure RPC public key and secret key pair. **chkey** prompts for the old secure-rpc password and verifies that it is correct by decrypting the secret key. If the user has not already keylogged in, **chkey** registers the secret key with the local **keyserv**(1M) daemon. If the secure-rpc password does not match the login password, **chkey** prompts for the login password. **chkey** uses the login password to encrypt the user’s secret Diffie-Hellman (192 bit) cryptographic key.

chkey ensures that the login password and the secure-rpc password are kept the same, thus enabling password shadowing, (see **shadow**(4)).

The key pair can be stored in the **/etc/publickey** file, (see **publickey**(4)), NIS **publickey** map or NIS+ **cred.org_dir** table. If a new secret key is generated, it will be registered with the local **keyserv**(1M) daemon.

If the source of the **publickey** is not specified with the –**s** option, **chkey** consults the **publickey** entry in the name service switch configuration file (see **nsswitch.conf**(4)). If the **publickey** entry specifies one and only one source, then **chkey** will change the key in the specified name service. However, if multiple name services are listed, **chkey** can not decide which source to update and will display an error message. The user should specify the source explicitly with the –**s** option.

Non root users are not allowed to change their key pair in the **files** database.

OPTIONS

–**p**	Re-encrypt the existing secret key with the user’s login password.
–**s nisplus**	Update the **NIS+** database.
–**s nis**	Update the **NIS** database.
–**s files**	Update the **files** database.

FILES

/etc/nsswitch.conf
/etc/publickey

SEE ALSO

keylogin(1), **keylogout**(1), **keyserv**(1M), **newkey**(1M), **nisaddcred**(1M), **nsswitch.conf**(4), **publickey**(4), **shadow**(4)

NAME

chmod – change the permissions mode of a file

SYNOPSIS

chmod [–**fR**] *mode filename*. . .
chmod [**ugoa**]{ **+** | **-** | **=** }[**rwxlsStTugo**] *filename*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

chmod changes or assigns the mode of a file. The mode of a file specifies its permissions and other attributes. The mode may be absolute or symbolic.

An absolute *mode* is specified using octal numbers:

chmod *nnnn filename* . . .

where *n* is a number from 0 to 7. An absolute mode is constructed from the OR of any of the following modes:

4000	Set user ID on execution.
20#0	Set group ID on execution if # is **7**, **5**, **3**, or **1**. Enable mandatory locking if # is **6**, **4**, **2**, or **0**. For directories, files are created with BSD semantics for propagation of the group ID. With this option, files inherit the group ID of the directory. It may be cleared only by using symbolic mode.
1000	Turn on sticky bit. See **chmod**(2).
0400	Allow read by owner.
0200	Allow write by owner.
0100	Allow execute (search in directory) by owner.
0070	Allow read, write, and execute (search) by group.
0007	Allow read, write, and execute (search) by others.

A symbolic *mode* is specified in the following format:

chmod [*who*] *operator* [*permission(s)*] *filename*. . .

who is zero or more of the characters **u**, **g**, **o**, and **a** specifying whose permissions are to be changed or assigned:

u	user's permissions
g	group's permissions
o	others' permissions
a	all permissions (user, group, and other)

If *who* is omitted, it defaults to **a**, but the setting of the file creation mask (see **umask** in **sh**(1) or **csh**(1) for more information) is taken into account. When *who* is omitted, **chmod** will not override the restrictions of your user mask.

operator is one of +, –, or =, signifying how permissions are to be changed:

+	Add permissions.
–	Take away permissions.
=	Assign permissions absolutely.

Unlike other symbolic operations, **=** has an absolute effect in that it resets all other bits. Omitting *permission*(s) is useful only with **=** to take away all permissions.

permission(s) is any compatible combination of the following letters:

r	read permission
w	write permission
x	execute permission
l	mandatory locking
s	user or group set-ID
S	undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
t	sticky bit
T	the 1000 bit is turned on, and execution is off (undefined bit-state)
u,**g**,**o**	indicate that *permission* is to be taken from the current user, group or other mode respectively.

Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	**rwx**	**rwx**

This example (user, group, and others all have permission to read, write, and execute a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously.

The letter **s** is only meaningful with **u** or **g**, and **t** only works with **u**.

Mandatory file and record locking (**l**) refers to a file's ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID bit and enable a file to be locked on execution at the same time. The following examples, therefore, are invalid and elicit error messages:

```
chmod g+x,+l filename
chmod g+s,+l filename
```

Only the owner of a file or directory (or the super-user) may change that file's or directory's mode. Only the super-user may set the sticky bit on a non-directory file. If you are not super-user, **chmod** will mask the sticky-bit but will not return an error. In order to turn on a file's set-group-ID bit, your own group ID must correspond to the file's and group execution must be set.

OPTIONS

–f Force. **chmod** will not complain if it fails to change the mode of a file.

–R Recursively descend through directory arguments, setting the mode for each file as described above. When symbolic links are encountered, the mode of the target file is changed, but no recursion takes place.

EXAMPLES

Deny execute permission to everyone:

```
example% chmod a–x filename
```

Allow read permission to everyone:

```
example% chmod 444 filename
```

Make a file readable and writable by the group and others:

```
example% chmod go+rw filename
example% chmod 066 filename
```

Cause a file to be locked during access:

```
example% chmod +l filename
```

Allow everyone to read, write, and execute the file and turn on the set group-ID.

```
example% chmod =rwx,g+s filename
example% chmod 2777 filename
```

Absolute changes don't work for the set-group-ID bit of a directory. You must use **g+s** or **g-s**.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **chmod** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **chmod** behaves.

LC_CTYPE

Determines how **chmod** handles characters. When **LC_CTYPE** is set to a valid value, **chmod** can display and handle text and filenames containing valid characters for that locale. **chmod** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **chmod** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

ls(1), **chmod**(2), **environ**(5)

NOTES

chmod permits you to produce useless modes so long as they are not illegal (for instance, making a text file executable). **chmod** does not check the file type to see if mandatory locking is meaningful.

If the filesystem is mounted with the *nosuid* option, *setuid* execution is not allowed.

NAME

chown – change owner of file

SYNOPSIS

chown [–**fhR**] *owner filename* ...

AVAILABILITY

SUNWcsu

DESCRIPTION

chown changes the owner of the *file*s to *owner*. The *owner* may be either a decimal user ID or a login name found in **/etc/passwd** file.

If **chown** is invoked by other than the super-user, the set-user-ID bit of the file mode, 04000, is cleared.

Only the owner of a file (or the super-user) may change the owner of that file.

The operating system has a configuration option {**_POSIX_CHOWN_RESTRICTED**}, to restrict ownership changes. When this option is in effect the owner of the file is prevented from changing the owner **ID** of the file. Only the super-user can arbitrarily change owner **ID**s whether this option is in effect or not.

OPTIONS

–**f** Do not report errors.

–**h** If the file is a symbolic link, change the owner of the symbolic link. Without this option, the owner of the file referenced by the symbolic link is changed.

–**R** Recursive. **chown** descends through the directory, and any subdirectories, setting the ownership ID as it proceeds. When a symbolic link is encountered, the owner of the target file is changed (unless the –**h** option is specified), but no recursion takes place.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **chown** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **chown** behaves.

LC_CTYPE

Determines how **chown** handles characters. When **LC_CTYPE** is set to a valid value, **chown** can display and handle text and filenames containing valid characters for that locale. **chown** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **chown** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/etc/passwd

SEE ALSO

chgrp(1), **chmod**(1), **chown**(2), **passwd**(4), **environ**(5)

NAME

chown – change owner

SYNOPSIS

/usr/ucb/chown [**–fR**] *owner*[.group] *filename* . . .

AVAILABILITY

SUNWscpu

DESCRIPTION

chown changes the owner of the *filename*s to *owner*. The owner may be either a decimal user ID (UID) or a login name found in the password file. An optional *group* may also be specified. The group may be either a decimal group ID (GID) or a group name found in the GID file.

Only the super-user can change owner, in order to simplify accounting procedures.

OPTIONS

–f	Do not report errors.
–R	Recursively descend into directories setting the ownership of all files in each directory encountered. When symbolic links are encountered, their ownership is changed, but they are not traversed.

FILES

/etc/passwd	password file

SEE ALSO

chgrp(1), **chown**(2), **group**(4), **passwd**(4)

NAME

ckdate, errdate, helpdate, valdate – prompts for and validates a date

SYNOPSIS

ckdate [–**Q**] [–**W** *width*] [–**f** *format*] [–**d** *default*] [–**h** *help*] [–**e** *error*]
[–**p** *prompt*] [–**k** *pid* [–**s** *signal*]]

/usr/sadm/bin/errdate [–**W** *width*] [–**e** *error*] [–**f** *format*]

/usr/sadm/bin/helpdate [–**W***width*] [–**h** *help*] [–**f** *format*]

/usr/sadm/bin/valdate [–**f** *format*] [*input*]

AVAILABILITY

SUNWcsu

DESCRIPTION

ckdate prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a date, text for help and error messages, and a default value (which will be returned if the user responds with a RETURN). The user response must match the defined format for a date.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The –**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) will be displayed.

Three visual tool modules are linked to the **ckdate** command. They are **errdate** (which formats and displays an error message), **helpdate** (which formats and displays a help message), and **valdate** (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt. When *format* is defined in the **errdate** and **helpdate** modules, the messages will describe the expected format.

OPTIONS

–**Q**	Specifies that quit will not be allowed as a valid response.
–**W** *width*	Specifies that prompt, help and error messages will be formatted to a line length of **width**.
–**f** *format*	Specifies the format against which the input will be verified. Possible formats and their definitions are:

%b = abbreviated month name (jan, feb, mar)
%B = full month name **%d** = day of month (01 - 31)
%D = date as %m / %d / %y (the default format)
%e = day of month (1 - 31; single digits are preceded by a blank)
%h = abbreviated month name, identical to **%b%**
%m = month number (01 - 12)
%y = year within century (for instance, 89)
%Y = year as **CCYY** (for instance, 1989)

–**d** *default*	Defines the default value as *default*. The default does not have to meet the format criteria.
–**h** *help*	Defines the help messages as *help*.
–**e** *error*	Defines the error message as *error*.
–**p** *prompt*	Defines the prompt message as *prompt*.
–**k** *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.
–**s** *signal*	Specifies that the process ID *pid* defined with the –**k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against format criteria.

EXIT CODES

0	Successful execution
1	EOF on input or negative width on –**W** option, or usage error
3	User termination (quit)
4	Garbled format argument

NOTES

The default prompt for **ckdate** is:

Enter the date [?,q]:

The default error message is:

ERROR - Please enter a date. Format is <*format*>.

The default help message is:

Please enter a date. Format is <*format*>.

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. The **valdate** module will not produce any output. It returns zero for success and non-zero for failure.

NAME

ckgid, errgid, helpgid, valgid – prompts for and validates a group id

SYNOPSIS

ckgid [–**Q**] [–**W** *width*] [–**m**] [–**d** *default*] [–**h** *help*] [–**e** *error*] [–**p** *prompt*]
[–**k** *pid* [–**s** *signal*]]

/usr/sadm/bin/errgid [–**W** *width*] [–**e** *error*]
/usr/sadm/bin/helpgid [–**W** *width*] [–**m**] [–**h** *help*]
/usr/sadm/bin/valgid *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

ckgid prompts a user and validates the response. It defines, among other things, a prompt message whose response should be an existing group ID, text for help and error messages, and a default value (which will be returned if the user responds with a carriage return).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The –**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) will be displayed.

Three visual tool modules are linked to the **ckgid** command. They are **errgid** (which formats and displays an error message), **helpgid** (which formats and displays a help message), and **valgid** (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt.

OPTIONS

–**Q**	Specifies that quit will not be allowed as a valid response.
–**W** *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
–**m**	Displays a list of all groups when help is requested or when the user makes an error.
–**d** *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
–**h** *help*	Defines the help messages as *help*.
–**e** *error*	Defines the error message as *error*.
–**p** *prompt*	Defines the prompt message as *prompt*.
–**k** *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.
–**s** *signal*	Specifies that the process ID *pid* defined with the –**k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against **/etc/group**.

EXIT CODES

0	Successful execution
1	EOF on input or negative width on −**W** option, or usage error
3	User termination (quit)

NOTES

The default prompt for **ckgid** is:

> **Enter the name of an existing group [?,q]:**

The default error message is:

> **ERROR: Please enter one of the following group names: [***List***]**

If the −**m** option of **ckgid** is used, a list of valid groups is displayed here.

The default help message is:

> **ERROR: Please enter one of the following group names: [***List***]**

If the −**m** option of **ckgid** is used, a list of valid groups is displayed here.

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. The **valgid** module will not produce any output. It returns zero for success and non-zero for failure.

NAME

ckint, errint, helpint, valint – display a prompt; verify and return an integer value

SYNOPSIS

ckint [−**Q**] [−**W** *width*] [−**b** *base*] [−**d** *default*] [−**h** *help*] [−**e** *error*] [−**p** *prompt*] [−**k** *pid* [−**s** *signal*]]

/usr/sadm/bin/errint [−**W** *width*] [−**b** *base*] [−**e** *error*]
/usr/sadm/bin/helpint [−**W** *width*] [−**b** *base*] [−**h** *help*]
/usr/sadm/bin/valint [−**b** *base*] *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

ckint prompts a user, then validates the response. It defines, among other things, a prompt message whose response should be an integer, text for help and error messages, and a default value (which will be returned if the user responds with a carriage return).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The −**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) will be displayed.

Three visual tool modules are linked to the **ckint** command. They are **errint** (which formats and displays an error message), **helpint** (which formats and displays a help message), and **valint** (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt. When *base* is defined in the **errint** and **helpint** modules, the messages will include the expected base of the input.

OPTIONS

−**Q**	Specifies that quit will not be allowed as a valid response.
−**W** *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
−**b** *base*	Defines the base for input. Must be **2** to **36**, default is **10**.
−**d** *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
−**h** *help*	Defines the help messages as *help*.
−**e** *error*	Defines the error message as *error*.
−**p** *prompt*	Defines the prompt message as *prompt*.
−**k** *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.

–s *signal*	Specifies that the process ID *pid* defined with the **–k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against *base* criterion.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on **–W** option, or usage error
3	User termination (quit)

NOTES

The default base 10 prompt for **ckint** is:

Enter an integer [?,q]:

The default base 10 error message is:

ERROR - Please enter an integer.

The default base 10 help message is:

Please enter an integer.

The messages are changed from ‘‘integer’’ to ‘‘base *base* integer’’ if the base is set to a number other than 10.

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. The **valint** module will not produce any output. It returns **0** for success and non-zero for failure.

NAME

ckitem, erritem, helpitem – build a menu; prompt for and return a menu item

SYNOPSIS

ckitem [–**Q**] [–**W** *width*] [–**uno**] [–**f** *filename*] [–**l** *label*] [[–**i** *invis*] [, ...]] [–**m** *max*] [–**d** *default*] [–**h** *help*] [–**e** *error*] [–**p** *prompt*] [–**k** *pid* [–**s** *signal*]] [*choice* [...]]

/usr/sadm/bin/erritem [–**W** *width*] [–**e** *error*] [*choice* [...]]

/usr/sadm/bin/helpitem [–**W** *width*] [–**h** *help*] [*choice* [...]]

AVAILABILITY

SUNWcsu

DESCRIPTION

ckitem builds a menu and prompts the user to choose one item from a menu of items. It then verifies the response. Options for this command define, among other things, a prompt message whose response will be a menu item, text for help and error messages, and a default value (which will be returned if the user responds with a carriage return).

By default, the menu is formatted so that each item is prepended by a number and is printed in columns across the terminal. Column length is determined by the longest choice. Items are alphabetized.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The –**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) will be displayed.

Two visual tool modules are linked to the **ckitem** command. They are **erritem** (which formats and displays an error message) and **helpitem** (which formats and displays a help message). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt. When *choice* is defined in these modules, the messages will describe the available menu choice (or choices).

OPTIONS

–**Q**	Specify that quit will not be allowed as a valid response.
–**W** *width*	Specify that prompt, help and error messages will be formatted to a line length of *width*.
–**u**	Specify that menu items should be displayed as an unnumbered list.
–**n**	Specify that menu items should not be displayed in alphabetical order.
–**o**	Specify that only one menu token will be returned.
–**f** *filename*	Define a file, *filename*, which contains a list of menu items to be displayed. (The format of this file is: **token<tab>description**. Lines beginning with a pound sign (#) are designated as comments and ignored.)
–**l** *label*	Define a label, *label*, to print above the menu.

–**i** *invis*	Define invisible menu choices (those which will not be printed in the menu). (For example, ‘‘all’’ used as an invisible choice would mean it is a legal option but does not appear in the menu. Any number of invisible choices may be defined.) Invisible choices should be made known to a user either in the prompt or in a help message.
–**m** *max*	Define the maximum number of menu choices that the user can choose. The default is 1.
–**d** *default*	Define the default value as *default*. The default is not validated and so does not have to meet any criteria.
–**h** *help*	Define the help messages as *help*.
–**e** *error*	Define the error message as *error*.
–**p** *prompt*	Define the prompt message as *prompt*.
–**k** *pid*	Specify that the process ID *pid* is to be sent a signal if the user chooses to abort.
–**s** *signal*	Specify that process ID *pid* defined with the –**k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
choice	Define menu items. Items should be separated by white space or newline.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on –**W** option, or inability to open file on –**f** option, or usage error
3	User termination (quit)
4	No choices from which to choose

NOTES

The user may input the number of the menu item if choices are numbered or as much of the string required for a unique identification of the item. Long menus are paged with 10 items per page.

When menu entries are defined both in a file (by using the –**f** option) and also on the command line, they are usually combined alphabetically. However, if the –**n** option is used to suppress alphabetical ordering, then the entries defined in the file are shown first, followed by the options defined on the command line.

The default prompt for **ckitem** is:

Enter selection [?,??,q]:

One question mark will give a help message and then redisplay the prompt. Two question marks will give a help message and then redisplay the menu label, the menu and the prompt.

The default error message if you typed a number is:

ERROR: Bad numeric choice specification

The default error message if you typed a string is:

> **ERROR: Entry does not match available menu selection. Enter the number of the menu item you wish to select, the token which is associated with the menu item, or a partial string which uniquely identifies the token for the menu item. Enter ?? to reprint the menu.**

The default help message is:

> **Enter the number of the menu item you wish to select, the token which is associated with the menu item, or a partial string which uniquely identifies the token for the menu item. Enter ?? to reprint the menu.**

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**.

NAME

ckkeywd – prompts for and validates a keyword

SYNOPSIS

ckkeywd [−**Q**] [−**W** *width*] [−**d** *default*] [−**h** *help*] [−**e** *error*] [−**p** *prompt*]
[−**k** *pid* [−**s** *signal*]] *keyword* [. . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

ckkeywd prompts a user and validates the response. It defines, among other things, a prompt message whose response should be one of a list of keywords, text for help and error messages, and a default value (which will be returned if the user responds with a carriage return). The answer returned from this command must match one of the defined list of keywords.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The −**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) will be displayed.

OPTIONS

−**Q**	Specifies that quit will not be allowed as a valid response.
−**W** *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
−**d** *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
−**h** *help*	Defines the help messages as *help*.
−**e** *error*	Defines the error message as *error*.
−**p** *prompt*	Defines the prompt message as *prompt*.
−**k** *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.
−**s** *signal*	Specifies that the process ID *pid* defined with the −**k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
keyword	Defines the keyword, or list of keywords, against which the answer will be verified.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on −**W** option, or no keywords from which to choose, or usage error

3	User termination (quit)

NOTES

The default prompt for **ckkeywd** is:

> **Enter appropriate value [***keyword***,[...],?,q]:**

The default error message is:

> **ERROR: Please enter one of the following keywords:**
> *keyword***,[...],q**

The default help message is:

> *keyword***,[...],q**

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**.

NAME

ckpath, errpath, helppath, valpath – display a prompt; verify and return a pathname

SYNOPSIS

ckpath [–**Q**] [–**W** *width*] [–**a** | **l**] [–**b** | **c** | **f** | **y**] [–**n** | [**o** | **z**]] [–**rtwx**]
[–**d** *default*] [–**h** *help*] [–**e** *error*] [–**p** *prompt*] [–**k** *pid* [–**s** *signal*]]

/usr/sadm/bin/errpath [–**W** *width*] [–**a** | **l**] [–**b** | **c** | **f** | **y**] [–**n** | [**o** | **z**]]
[–**rtwx**] [–**e** *error*]

/usr/sadm/bin/helppath [–**W** *width*] [–**a** | **l**] [–**b** | **c** | **f** | **y**] [–**n** | [**o** | **z**]]
[–**rtwx**] [–**h** *help*]

/usr/sadm/bin/valpath [–**a** | **l**] [–**b** | **c** | **f** | **y**] [–**n** | [**o** | **z**]] [–**rtwx**] *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

ckpath prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a pathname, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

The pathname must obey the criteria specified by the first group of options. If no criteria is defined, the pathname must be for a normal file that does not yet exist. If neither –**a** (absolute) or –**l** (relative) is given, then either is assumed to be valid.

All messages are limited in length to 79 characters and are formatted automatically. Tabs and newlines are removed after a single white space character in a message definition, but spaces are not removed. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **EXAMPLES**) is displayed.

Three visual tool modules are linked to the **ckpath** command. They are **errpath** (which formats and displays an error message on the standard output), **helppath** (which formats and displays a help message on the standard output), and **valpath** (which validates a response). These modules should be used in conjunction with Framed Access Command Environment (FACE) objects. In this instance, the FACE object defines the prompt.

OPTIONS

–**Q**	Specify that quit is not allowed as a valid response.
–**W** *width*	Specify that prompt, help and error messages be formatted to a line length of *width*.
–**a**	Pathname must be an absolute path.
–**l**	Pathname must be a relative path.
–**b**	Pathname must be a block special file.
–**c**	Pathname must be a character special file.
–**f**	Pathname must be a regular file.
–**y**	Pathname must be a directory.

−n	Pathname must not exist (must be new).
−o	Pathname must exist (must be old).
−z	Pathname must have a file having a size greater than 0 bytes.
−r	Pathname must be readable.
−t	Pathname must be creatable (touchable). Pathname will be created if it does not already exist.
−w	Pathname must be writable.
−x	Pathname must be executable.
−d *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
−h *help*	Defines the help message as *help*.
−e *error*	Defines the error message as *error*.
−p *prompt*	Defines the prompt message as *prompt*.
−k *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to quit.
−s *signal*	Specifies that the process ID *pid* defined with the **−k** option is to be sent signal *signal* when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against validation options.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on **−W** option, or usage error
2	Mutually exclusive options
3	User termination (quit)
4	Mutually exclusive options

EXAMPLES

The text of the default messages for **ckpath** depends upon the criteria options that have been used. An example default prompt for **ckpath** (using the **−a** option) is:

```
example% ckpath −a
Enter an absolute pathname [?,q]
```

An example default error message (using the **−a** option) is:

```
example% /usr/sadm/bin/errpath −a
ERROR: A pathname is a filename, optionally  preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

An example default help message (using the **−a** option) is:

```
example% /usr/sadm/bin/helppath −a
A pathname is a filename, optionally  preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. Quit input gets a trailing newline.

The **valpath** module will produce a usage message on stderr. It returns **0** for success and non-zero for failure.

```
example% /usr/sadm/bin/valpath
usage: valpath [–[a | l][b | c | f | y][n | [o | z]]rtwx] input
        .
        .
        .
```

SEE ALSO **face**(1), **signal**(5)

NAME

ckrange, errange, helprange, valrange – prompts for and validates an integer

SYNOPSIS

ckrange [**–Q**] [**–W** *width*] [**–l** *lower*] [**–u** *upper*] [**–b** *base*] [**–d** *default*] [**–h** *help*] [**–e** *error*] [**–p** *prompt*] [**–k** *pid* [**–s** *signal*]]

/usr/sadm/bin/errange [**–W** *width*] [**–e** *error*] [**–l** *lower*] [**–u** *upper*] [**–b** *base*]

/usr/sadm/bin/helprange [**–W** *width*] [**–h** *help*] [**–l** *lower*] [**–u** *upper*] [**–b** *base*]

/usr/sadm/bin/valrange [**–l** *lower*] [**–u** *upper*] [**–b** *base*] *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

ckrange prompts a user for an integer between a specified range and determines whether this response is valid. It defines, among other things, a prompt message whose response should be an integer in the range specified, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

This command also defines a range for valid input. If either the lower or upper limit is left undefined, then the range is bounded on only one end.

All messages are limited in length to 79 characters and are formatted automatically. Tabs and newlines are removed after a single whitespace character in a message definition, but spaces are not removed. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **EXAMPLES**) is displayed.

Three visual tool modules are linked to the **ckrange** command. They are **errange** (which formats and displays an error message on the standard output), **helprange** (which formats and displays a help message on the standard output), and **valrange** (which validates a response). These modules should be used in conjunction with Framed Access Command Environment (FACE) objects. In this instance, the FACE object defines the prompt.

Note: Negative "input" arguments confuse **getopt** in **valrange**. By inserting a "**--**" before the argument, **getopt** processing will stop. See **getopt**(1) and **intro**(1) about **getopt** parameter handling. **getopt** is used to parse positional parameters and to check for legal options.

OPTIONS

–Q	Specifies that quit will not be allowed as a valid response.
–W *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
–l *lower*	Defines the lower limit of the range as *lower*. Default is the machine’s largest negative long.

–u *upper*	Defines the upper limit of the range as *upper*. Default is the machine's largest positive long.
–b *base*	Defines the base for input. Must be 2 to 36, default is 10. Base conversion uses **strtol**(3C). Output is always base 10.
–d *default*	Defines the default value as *default*. *default* is converted using **strtol**(3C) in the desired base. Any characters invalid in the specified base will terminate the **strtol** conversion without error.
–h *help*	Defines the help message as *help*.
–e *error*	Defines the error message as *error*.
–p *prompt*	Defines the prompt message as *prompt*.
–k *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to quit.
–s *signal*	Specifies that the process ID *pid* defined with the **–k** option is to be sent signal *signal* when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against upper and lower limits and base.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on **–W** option, or usage error
2	Usage error
3	User termination (quit)

EXAMPLES

The default base 10 prompt for **ckrange** is:

```
example% ckrange
Enter an integer between lower_bound and
upper_bound [lower_bound–upper_bound,?,q]:
```

The default base 10 error message is:

```
example% /usr/sadm/bin/errange
ERROR: Please enter an integer between lower_bound and upper_bound.
```

The default base 10 help message is:

```
example% /usr/sadm/bin/helprange
Please enter an integer between lower_bound and upper_bound.
```

The messages are changed from ‘‘integer’’ to ‘‘base *base* integer’’ if the base is set to a number other than 10, for example, **example% /usr/sadm/bin/helprange –b 36**.

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. Quit input gets a trailing newline.

The **valrange** module will produce a usage message on stderr. It returns **0** for success and non-zero for failure.

```
example% /usr/sadm/bin/valrange
usage: valrange [–l lower] [–u upper] [–b base] input
```

SEE ALSO

intro(1), **face**(1), **getopt**(1), **strtol**(3C), **signal**(5)

NAME

ckstr, errstr, helpstr, valstr – display a prompt; verify and return a string answer

SYNOPSIS

ckstr [**–Q**] [**–W** *width*] [[**–r** *regexp*] [. . .]] [**–l** *length*] [**–d** *default*] [**–h** *help*] [**–e** *error*] [**–p** *prompt*] [**–k** *pid* [**–s** *signal*]]

/usr/sadm/bin/errstr [**–W** *width*] [**–e** *error*] [**–l** *length*] [[**–r** *regexp*] [. . .]]

/usr/sadm/bin/helpstr [**–W** *width*] [**–h** *help*] [**–l** *length*] [[**–r** *regexp*] [. . .]]

/usr/sadm/bin/valstr [**–l** *length*] [[**–r** *regexp*] [. . .]] *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

ckstr prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a string, text for help and error messages, and a default value (which are returned if the user responds with a RETURN).

The answer returned from this command must match the defined regular expression and be no longer than the length specified. If no regular expression is given, valid input must be a string with a length less than or equal to the length defined with no internal, leading or trailing white space. If no length is defined, the length is not checked.

All messages are limited in length to 79 characters and are formatted automatically. Tabs and newlines are removed after a single white space character in a message definition, but spaces are not removed. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **EXAMPLES**) is displayed.

Three visual tool modules are linked to the **ckstr** command. They are **errstr** (which formats and displays an error message on the standard output), **helpstr** (which formats and displays a help message on the standard output), and **valstr** (which validates a response). These modules should be used in conjunction with Framed Access Command Environment (FACE) objects. In this instance, the FACE object defines the prompt.

OPTIONS

–Q	Specifies that quit will not be allowed as a valid response.
–W *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
–r *regexp*	Specifies a regular expression, **regexp**, against which the input should be validated. May include white space. If multiple expressions are defined, the answer need match only one of them.
–l *length*	Specifies the maximum length of the input.
–d *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
–h *help*	Defines the help message as *help*.
–e *error*	Defines the error message as *error*.

–**p** *prompt*	Defines the prompt message as *prompt*.
–**k** *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to quit.
–**s** *signal*	Specifies that the process ID *pid* defined with the –**k** option is to be sent signal *signal* when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against format length and / or regular expression criteria.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on –**W** option, or usage error
2	Invalid regular expression
3	User termination (quit)

EXAMPLES

The default prompt for **ckstr** is:

```
example% ckstr
Enter an appropriate value [?,q]:
```

The default error message is dependent upon the type of validation involved. The user will be told either that the length or the pattern matching failed. The default error message is:

```
example% /usr/sadm/bin/errstr
ERROR: Please enter a string which contains no embedded,
leading or trailing spaces or tabs.
```

The default help message is also dependent upon the type of validation involved. If a regular expression has been defined, the message is:

```
example% /usr/sadm/bin/helpstr –r regexp
Please enter a string which matches the following pattern:
regexp
```

Other messages define the length requirement and the definition of a string.

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. Quit input gets a trailing newline.

The **valstr** module will produce a usage message on stderr. It returns **0** for success and non-zero for failure.

```
example% /usr/sadm/bin/valstr
usage: valstr [–l length] [[–r regexp] [ . . . ]] input
```

SEE ALSO

face(1), **signal**(5)

NAME

cktime, errtime, helptime, valtime – display a prompt; verify and return a time of day

SYNOPSIS

cktime [–**Q**] [–**W** *width*] [–**f** *format*] [–**d** *default*] [–**h** *help*] [–**e** *error*] [–**p** *prompt*] [–**k** *pid* [–**s** *signal*]]

/usr/sadm/bin/errtime [–**W** *width*] [–**e** *error*] [–**f** *format*]

/usr/sadm/bin/helptime [–**W** *width*] [–**h** *help*] [–**f** *format*]

/usr/sadm/bin/valtime [–**f** *format*] *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

cktime prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a time, text for help and error messages, and a default value (which is returned if the user responds with a RETURN). The user response must match the defined format for the time of day.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including NEWLINE) is stripped. The –**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) is displayed.

Three visual tool modules are linked to the **cktime** command. They are **errtime** (which formats and displays an error message), **helptime** (which formats and displays a help message), and **valtime** (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt. When *format* is defined in the **errtime** and **helptime** modules, the messages will describe the expected format.

OPTIONS

–**Q**	Specifies that quit will not be allowed as a valid response.
–**W** *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
–**f** *format*	Specifies the format against which the input will be verified. Possible formats and their definitions are:

%H = hour (00 - 23)
%I = hour (00 - 12)
%M = minute (00 - 59)
%p = ante meridian or post meridian
%r = time as **%I:%M:%S %p**
%R = time as **%H:%M** (the default format)
%S = seconds (00 - 59)
%T = time as **%H:%M:%S**

−d *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
−h *help*	Defines the help messages as *help*.
−e *error*	Defines the error message as *error*.
−p *prompt*	Defines the prompt message as *prompt*.
−k *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.
−s *signal*	Specifies that the process ID *pid* defined with the **−k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against format criteria.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on **−W** option, or usage error
3	User termination (quit)
4	Garbled format argument

NOTES

The default prompt for **cktime** is:

> **Enter a time of day [?,q]:**

The default error message is:

> **ERROR: Please enter the time of day. Format is** *<format>***.**

The default help message is:

> **Please enter the time of day. Format is** *<format>***.**

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. The **valtime** module will not produce any output. It returns **0** for success and non-zero for failure.

NAME

ckuid, erruid, helpuid, valuid – prompts for and validates a user ID

SYNOPSIS

ckuid [–**Q**] [–**W** *width*] [–**m**] [–**d** *default*] [–**h** *help*] [–**e** *error*] [–**p** *prompt*]
[–**k** *pid* [–**s** *signal*]]
/usr/sadm/bin/erruid [–**W** *width*] [–**e** *error*]
/usr/sadm/bin/helpuid [–**W** *width*] [–**m**] [–**h** *help*]
/usr/sadm/bin/valuid *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

ckuid prompts a user and validates the response. It defines, among other things, a prompt message whose response should be an existing user ID, text for help and error messages, and a default value (which are returned if the user responds with a RETURN).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including NEWLINE) is stripped. The –**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) is displayed.

Three visual tool modules are linked to the **ckuid** command. They are **erruid** (which formats and displays an error message), **helpuid** (which formats and displays a help message), and **valuid** (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt.

OPTIONS

–**Q**	Specifies that quit will not be allowed as a valid response.
–**W** *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
–**m**	Displays a list of all logins when help is requested or when the user makes an error.
–**d** *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
–**h** *help*	Defines the help messages as *help*.
–**e** *error*	Defines the error message as *error*.
–**p** *prompt*	Defines the prompt message as *prompt*.
–**k** *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.
–**s** *signal*	Specifies that the process ID *pid* defined with the –**k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified against **/etc/passwd**.

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on –**W** option, or usage error
2	Usage error
3	User termination (quit)

NOTES

The default prompt for **ckuid** is:

Enter the login name of an existing user [?,q]:

The default error message is:

ERROR - Please enter the login name of an existing user.

If the –**m** option is used, the default error message is:

ERROR: Please enter one of the following login names: <List>

The default help message is:

Please enter the login name of an existing user.

If the –**m** option is used, the default help message is:

Please enter one of the following login names: <List>

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. The **valuid** module will not produce any output. It returns **0** for success and non-zero for failure.

NAME

ckyorn, erryorn, helpyorn, valyorn – prompts for and validates yes / no

SYNOPSIS

ckyorn [–**Q**] [–**W** *width*] [–**d** *default*] [–**h** *help*] [–**e** *error*] [–**p** *prompt*]
[–**k** *pid* [–**s** *signal*]]

/usr/sadm/bin/erryorn [–**W** *width*] [–**e** *error*]

/usr/sadm/bin/helpyorn [–**W** *width*] [–**h** *help*]

/usr/sadm/bin/valyorn *input*

AVAILABILITY

SUNWcsu

DESCRIPTION

ckyorn prompts a user and validates the response. It defines, among other things, a prompt message for a yes or no answer, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The –**W** option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under **NOTES**) is displayed.

Three visual tool modules are linked to the **ckyorn** command. They are **erryorn** (which formats and displays an error message), **helpyorn** (which formats and displays a help message), and **valyorn** (which validates a response). These modules should be used in conjunction with FACE objects. In this instance, the FACE object defines the prompt.

OPTIONS

–**Q**	Specifies that quit will not be allowed as a valid response.
–**W** *width*	Specifies that prompt, help and error messages will be formatted to a line length of *width*.
–**d** *default*	Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.
–**h** *help*	Defines the help messages as *help*.
–**e** *error*	Defines the error message as *error*.
–**p** *prompt*	Defines the prompt message as *prompt*.
–**k** *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.
–**s** *signal*	Specifies that the process ID *pid* defined with the –**k** option is to be sent signal **signal** when quit is chosen. If no signal is specified, **SIGTERM** is used.
input	Input to be verified as **y**, **yes**, or **n**, **no** (in any combination of upper- and lower-case letters).

EXIT CODES

0	Successful execution
1	EOF on input, or negative width on **–W** option, or usage error
2	Usage error
3	User termination (quit)

NOTES

The default prompt for **ckyorn** is:

> **Yes or No [y,n,?,q]:**

The default error message is:

> **ERROR - Please enter yes or no.**

The default help message is:

> **To respond in the affirmative, enter y, yes, Y, or YES.**
> **To respond in the negative, enter n, no, N, or NO.**

When the quit option is chosen (and allowed), **q** is returned along with the return code **3**. The **valyorn** module will not produce any output. It returns **0** for success and non-zero for failure.

NAME

clear – clear the terminal screen

SYNOPSIS

clear

AVAILABILITY

SUNWcsu

DESCRIPTION

clear clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen.

NAME

cmp – compare two files

SYNOPSIS

cmp [**–l**] [**–s**] *filename1 filename2* [*skip1*] [*skip2*]

AVAILABILITY

SUNWcsu

DESCRIPTION

The two files are compared. (If *filename1* is –, the standard input is used.) Under default options, **cmp** makes no comment if the files are the same; if they differ, it announces the byte and line numbers at which the first difference occurred. If one file is an initial subsequence of the other, that fact is noted. *skip1* and *skip2* are initial byte offsets into *filename1* and *filename2* respectively, and may be either octal or decimal; a leading 0 denotes octal.

OPTIONS

–l Print the byte number (decimal) and the differing bytes (octal) for each difference.

–s Print nothing for differing files; return exit codes only.

EXAMPLE

The following example:

```
example% cmp file1 file2 0 1024
```

does a byte for byte comparison of *file1* and *file2*. It skips the first 1024 bytes in *file2* before starting the comparison.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **cmp** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **cmp** behaves.

LC_CTYPE

Determines how **cmp** handles characters. When **LC_CTYPE** is set to a valid value, **cmp** can display and handle text and filenames containing valid characters for that locale. **cmp** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **cmp** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

comm(1), **diff**(1), **environ**(5)

DIAGNOSTICS

Exit code **0** is returned for identical files, **1** for different files, and **2** for an inaccessible or missing argument.

NAME

cof2elf – COFF to ELF object file translation

SYNOPSIS

cof2elf [–**iqV**] [–**Qy** | **n**] [–**s** *directory*] *files*

AVAILABILITY

x86

SUNWbtool

DESCRIPTION

cof2elf converts one or more COFF object *files* to ELF. This translation occurs in place, which means that the original file contents are modified. If an input file is an archive, each member will be translated as necessary, and the archive will be rebuilt with its members in the original order. **cof2elf** does not change input files that are not COFF.

Options have the following meanings:

–**i**	Normally, the files are modified only when full translation occurs. Unrecognized data, such as unknown relocation types, are treated as errors and prevent translation. Using the –**i** flag causes the program to ignore these partial translation conditions and modify the file anyway.
–**q**	Normally, **cof2elf** prints a message for each file it examines, reporting whether the file was translated, ignored, etc. Using the –**q** (quiet) flag suppresses these messages.
–**Q***arg*	If *arg* is **y**, identification information about **cof2elf** will be added to the output files. This can be useful for software administration. Giving **n** for *arg* explicitly asks for no such information, which is the default behavior.
–**s** *directory*	As mentioned above, **cof2elf** modifies the input files. This option saves a copy of the original files in the specified *directory*, which must already exist. **cof2elf** does not save files it does not modify.
–**V**	This flag tells **cof2elf** to print a version message on standard error.

SEE ALSO

ld(1), **elf**(3E), **a.out**(4), **ar**(4)

NOTES

Some debugging information is discarded. Although this does not affect the behavior of a running program, it may affect the information available for symbolic debugging.

cof2elf translates only COFF relocatable files. It does not translate executable or static shared library files for two main reasons. First, the operating system supports executable files and static shared libraries, making translation unnecessary. Second, those files have specific address and alignment constraints determined by the file format. Matching the constraints with a different object file format is problematic.

When possible, programmers should recompile their source code to build new object files. **cof2elf** is provided for those times when source code is unavailable.

NAME

col – reverse line-feeds filter

SYNOPSIS

col [–**b**] [–**f**] [–**x**] [–**p**]

AVAILABILITY

SUNWesu

DESCRIPTION

col reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-**7**), and by forward and reverse half-line-feeds (ESC-**9** and ESC-**8**). **col** is particularly useful for filtering multicolumn output made with the **.rt** command of **nroff** and output resulting from use of the **tbl**(1) preprocessor.

The ASCII control characters SO (\017) and SI (\016) are assumed by **col** to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output SI and SO characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, SI, SO, VT (\013), and ESC followed by **7**, **8**, or **9**. The VT character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

OPTIONS

–**b** **col** assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

–**f** Although **col** accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the –**f** (fine) option; in this case, the output from **col** may contain forward half-line-feeds (ESC-**9**), but will still never contain either kind of reverse line motion.

–**x** Unless the –**x** option is given, **col** will convert white space to tabs on output wherever possible to shorten printing time.

–**p** Normally, **col** will ignore any escape sequences unknown to it that are found in its input; the –**p** option may be used to cause **col** to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **col** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **col** behaves.

LC_CTYPE
Determines how **col** handles characters. When **LC_CTYPE** is set to a valid value, **col** can display and handle text and filenames containing valid characters for that locale. **col** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **col** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

SEE ALSO

nroff(1), **tbl**(1), **ascii**(5), **environ**(5)

NOTES

The input format accepted by **col** matches the output produced by **nroff** with either the –**T37** or –**Tlp** options. Use –**T37** (and the –**f** option of **col**) if the ultimate disposition of the output of **col** will be a device that can interpret half-line motions, and –**Tlp** otherwise.

col cannot back up more than 128 lines or handle more than 800 characters per line.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [**–1** | **–2** | **–3** | **–12** | **–13** | **–23**] *filename1 filename2*

AVAILABILITY

SUNWesu

DESCRIPTION

comm reads *filename1* and *filename2*, which should be ordered in ASCII collating sequence (see **sort**(1)), and produces a three-column output: lines only in *filename1*; lines only in *filename2*; and lines in both files. The file name – means the standard input.

OPTIONS

The following options can be used to suppress the indicated columns from the display.

–1	Suppress column 1; omit lines only in *filename1*.
–2	Suppress column 2; omit lines only in *filename2*.
–3	Suppress column 3; omit lines common to both files.
–12	Suppress columns 1 and 2; only show lines common to both files.
–13	Suppress columns 1 and 3; only show lines in *filename2*.
–23	Suppress columns 2 and 3; only show lines in *filename1*.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **comm** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **comm** behaves.

LC_CTYPE

Determines how **comm** handles characters. When **LC_CTYPE** is set to a valid value, **comm** can display and handle text and filenames containing valid characters for that locale. **comm** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **comm** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

cmp(1), **diff**(1), **sort**(1), **uniq**(1), **environ**(5)

NOTES

You can specify **–123**, but doing so suppresses all output.

NAME

compress, uncompress, zcat – compress, uncompress files or display expanded files

SYNOPSIS

compress [**–cfv**] [**–b** *bits*] [*filename*. . .]

uncompress [**–cfv**] [*filename*. . .]

zcat [*filename*. . .]

AVAILABILITY

SUNWesu

DESCRIPTION

compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with a **.Z**, extension. The ownership modes (if super-user), access time and modification time will stay the same. If no files are specified, the standard input is compressed to the standard output.

The amount of compression obtained depends on the size of the input, the number of *bits* per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50–60%. Compression is generally much better than that achieved by Huffman coding (as used in **pack**(1)), and takes less time to compute. The *bits* parameter specified during compression is encoded within the compressed file, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is subsequently allowed.

Compressed files can be restored to their original form using **uncompress**.

zcat produces uncompressed output on the standard output, but leaves the compressed **.Z** file intact.

OPTIONS

–c Write to the standard output; no files are changed. The nondestructive behavior of **zcat** is identical to that of '**uncompress –c**'.

–f When compressing, force compression, even if the file does not actually shrink, or the corresponding **.Z** file already exists. Except when running in the background (under **/usr/bin/sh**), if **–f** is not given, prompt to verify whether an existing **.Z** file should be overwritten. When uncompressing, forces uncompression, even if **filename** exists.

–v Verbose. Display the percentage reduction for each file compressed.

–b *bits* Set the upper limit (in bits) for common substring codes. *bits* must be between 9 and 16 (16 is the default). Lowering the number of bits will result in larger, less compressed files.

FILES

/usr/bin/sh

SEE ALSO

pack(1)

DIAGNOSTICS

Exit status is normally **0**. If the last file was not compressed because it became larger, the status is **2**. If an error occurs, exit status is **1**.

Usage: compress [–fvc] [–b maxbits] [*filename* . . .**]**
Invalid options were specified on the command line.

Missing maxbits
Maxbits must follow **–b**.

filename: **not in compressed format**
The file specified to **uncompress** has not been compressed.

filename: **compressed with** *xx***bits, can only handle** *yy***bits**
filename was compressed by a program that could deal with more *bits* than the compress code on this machine. Recompress the file with smaller *bits*.

filename: **already has .Z suffix -- no change**
The file is assumed to be already compressed. Rename the file and try again.

filename: **already exists; do you wish to overwrite (y or n)?**
Respond **y** if you want the output file to be replaced; **n** if not.

uncompress: corrupt input
A **SIGSEGV** violation was detected, which usually means that the input file is corrupted.

Compression: *xx.xx***%**
Percentage of the input saved by compression. (Relevant only for **–v**.)

– – not a regular file: unchanged
When the input file is not a regular file, (such as a directory), it is left unaltered.

– – has *xx* **other links: unchanged**
The input file has links; it is left unchanged. See **ln**(1) for more information.

– – file unchanged
No savings are achieved by compression. The input remains uncompressed.

NOTES

Although compressed files are compatible between machines with large memory, **–b** 12 should be used for file transfer to architectures with a small process data space (64KB or less).

compress should be more flexible about the existence of the **.Z** suffix.

NAME

coproc, cocreate, cosend, cocheck, coreceive, codestroy – communicate with a process

SYNOPSIS

cocreate [**–r** *rpath*] [**–w** *wpath*] [**–i** *id*] [**–R** *refname*] [**–s** *send_string*]
[**–e** *expect_string*] *command*

cosend [**–n**] *proc_id string*

cocheck *proc_id*

coreceive *proc_id*

codestroy [**–R** *refname*] *proc_id* [*string*]

DESCRIPTION

These co-processing functions provide a flexible means of interaction between FMLI and an independent process; especially, they enable FMLI to be responsive to asynchronous activity.

The **cocreate** function starts *command* as a co-process and initializes communications by setting up pipes between FMLI and the standard input and standard output of *command.* The argument *command* must be an executable and its arguments (if any). This means that *command* expects strings on its input (supplied by **cosend**) and sends information on its output that can be handled in various ways by FMLI.

The **cosend** function sends *string* to the co-process identified by *proc_id* via the pipe set up by **cocreate** (optionally *wpath*), where *proc_id* can be either the *command* or *id* specified in **cocreate**. By default, **cosend** blocks, waiting for a response from the co-process. Also by default, FMLI does not send a *send_string* and does not expect an *expect_string* (except a newline). That is, it reads only one line of output from the co-process. If **–e** *expect_string* was not defined when the pipe was created, then the output of the co-process is any single string followed by a newline: any other lines of output remain on the pipe. If the **–e** option was specified when the pipe was created, **cosend** reads lines from the pipe until it reads a line starting with *expect_string.* All lines except the line starting with *expect_string* become the output of **cosend**.

The **cocheck** function determines if input is available from the process identified by *proc_id*, where *proc_id* can be either the *command* or *id* specified in **cocreate**. It returns a Boolean value, which makes **cocheck** useful in **if** statements and in other backquoted expressions in Boolean descriptors. **cocheck** receives no input from the co-process; it simply indicates if input is available from the co-process. You must use **coreceive** to actually accept the input. The **cocheck** function can be called from a **reread** descriptor to force a frame to update when new data is available. This is useful when the default value of a field in a form includes **coreceive**.

The **coreceive** function is used to read input from the co-process identified by *proc_id*, where *proc_id* can be either the *command* or *id* specified in **cocreate**. It should only be used when it has been determined, using **cocheck**, that input is actually available. If the **–e** option was used when the co-process was created, **coreceive** will continue to return lines of input until *expect_string* is read. At this point, **coreceive** will terminate. The output of **coreceive** is all the lines that were read excluding the line starting with *expect_string.* If the **–e** option was not used in the **cocreate**, each invocation of **coreceive** will return exactly one line from the co-process. If no input is available when **coreceive** is invoked, it will simply terminate without producing output.

The **codestroy** function terminates the read / write pipes to *proc-id*, where *proc_id* can be either the *command* or *id* specified in **cocreate**. It generates a **SIGPIPE** signal to the (child) co-process. This kills the co-process, unless the co-process ignores the **SIGPIPE** signal. If the co-process ignores the **SIGPIPE**, it will not die, even after the FMLI process terminates (the parent process id of the co-process will be **1**).

The optional argument *string* is sent to the co-process before the co-process dies. If *string* is not supplied, a NULL string is passed, followed by the normal *send_string* (newline by default). That is, **codestroy** will call **cosend** *proc_id string*: this implies that **codestroy** will write any output generated by the co-process to *stdout*. For example, if an interactive co-process is written to expect a "quit" string when the communication is over, the **close** descriptor could be defined;

```
close=`codestroy ID ’quit’ | message`
```

and any output generated by the co-process when the string **quit** is sent to it via **codestroy** (using **cosend**) would be redirected to the message line.

The **codestroy** function should usually be given the −**R** option, since you may have more than one process with the same name, and you do not want to kill the wrong one. **codestroy** keeps track of the number of *refnames* you have assigned to a process with **cocreate**, and when the last instance is killed, it kills the process (*id*) for you. **codestroy** is typically called as part of a **close** descriptor because **close** is evaluated when a frame is closed. This is important because the co-process will continue to run if **codestroy** is not issued.

When writing programs to use as co-processes, the following tips may be useful. If the co-process program is written in C language, be sure to flush output after writing to the pipe. (Currently, **awk**(1) and **sed**(1) cannot be used in a co-process program because they do not flush after lines of output.) Shell scripts are well-mannered, but slow. C language is recommended. If possible, use the default *send_string*, *rpath* and *wpath*. In most cases, *expect_string* will have to be specified. This, of course, depends on the co-process.

In the case where asynchronous communication from a co-process is desired, a co-process program should use **vsig** to force strings into the pipe and then signal FMLI that output from the co-process is available. This causes the **reread** descriptor of all frames to be evaluated immediately.

OPTIONS

cocreate options are:

–**r** *rpath*	If –**r** is specified, *rpath* is the pathname from which FMLI reads information. This option is usually used to set up communication with processes that naturally write to a certain path. If –**r** is not specified, **cocreate** will choose a unique path in **/var/tmp**.
–**w** *wpath*	If –**w** is specified, *wpath* is the pathname to which **cosend** writes information. This option is usually used so that one process can talk to many different FMLI processes through the same pipe. If –**w** is not specified, **cocreate** will choose a unique path in **/var/tmp**.
–**i** *id*	If –**i** is specified, *id* is an alternative name for the co-processinitialized by this **cocreate**. If –**i** is not specified, *id* defaults to *command*. The argument *id* can later be used with the other co-processing functions rather than *command*. This option is typically used, since it facilitates the creation of two or more co-processes generated from the same *command*. (For example, **cocreate -i ID1 program args** and **cocreate -i ID2 program different_args**).
–**R** *refname*	If –**R** is specified, *refname* is a local name for the co-process. Since the **cocreate** function can be issued more than once, a *refname* is useful when the same co-process is referenced a second or subsequent time. With the –**R** option, if the co-process already exists a new one will not be created: the same pipes will be shared. Then, *refname* can be used as an argument to the –**R** option to **codestroy** when you want to end a particular connection to a co-process and leave other connections undisturbed. (The co-process is only killed after **codestroy** –**R** has been called as many times as **cocreate** –**R** was called.)
–**s** *send_string*	The –**s** option specifies *send_string* as a string that will be appended to all output sent to the co-process using **cosend**. This option allows a co-process to know when input from FMLI has completed. The default *send_string* is a newline if –**s** is not specified.
–**e** *expect_string*	The –**e** option specifies *expect_string* as a string that identifies the end of all output returned by the co-process. (Note: *expect_string* need only be the initial part of a line, and there must be a newline at the end of the co-process output.) This option allows FMLI to know when output from the co-process has completed. The default *expect_string* is a newline if –**e** is not specified.

cosend options are:

–**n**	If the –**n** option is specified, **cosend** will not wait for a response from the co-process. It simply returns, providing no output. If the –**n** option is not used, a co-process that does not answer will cause FMLI to permanently hang, waiting for input from the co-process.

EXAMPLES

```
   .
   .
   .
init=`cocreate –i BIGPROCESS initialize`
close=`codestroy BIGPROCESS`
   .
   .
   .
reread=`cocheck BIGPROCESS`

name=`cosend –n BIGPROCESS field1`
   .
   .
   .
name="Receive field"
inactive=TRUE
value=`coreceive BIGPROCESS`
```

SEE ALSO

awk(1), **cat**(1), **sed**(1), **vsig**(1F)

NOTES

If **cosend** is used without the –**n** option, a co-process that does not answer will cause FMLI to permanently hang.

The use of non-alphabetic characters in input and output strings to a co-process should be avoided because they may not get transferred correctly.

NAME

cp – copy files

SYNOPSIS

cp [–**i**] [–**p**] [–**r**] [*filename* . . .] *target*

AVAILABILITY

SUNWcsu

DESCRIPTION

The **cp** command copies *filename* to *target*. *filename* and *target* cannot have the same name. Be careful when using **sh**(1) metacharacters. If *target* is not a directory, only one file may be specified before it; if it is a directory, more than one file may be specified. If *target* does not exist, **cp** creates a file named *target*. If *target* exists and is not a directory, its contents are overwritten. If *target* is a directory, the file(s) are copied to that directory. *target* and *filename* do not have to share the same parent directory.

If *filename* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

If *target* does not exist, **cp** creates a new file named *target* which has the same mode as *filename* except that the sticky bit is not set unless the user is a privileged user; the owner and group of *target* are those of the user.

If *target* is a file, its contents are overwritten, but the mode, owner, and group associated with it are not changed. The last modification time of *target* and the last access time of *filename* are set to the time the copy was made.

If *target* is a directory, then for each file named, a new file with the same mode is created in the target directory; the owner and the group are those of the user making the copy.

OPTIONS

–**i**	Interactive. **cp** will prompt for confirmation whenever the copy would overwrite an existing *target*. A **y** answer means that the copy should proceed. Any other answer prevents **cp** from overwriting *target*.
–**p**	Preserve. **cp** will duplicate not only the contents of *filename*, but also preserves the modification time and permission modes.
–**r**	If *filename* is a directory, **cp** will copy the directory and all its files, including any subdirectories and their files; *target* must be a directory.

EXAMPLES

To copy a file:

```
example% cp goodies goodies.old
example% ls goodies*
goodies goodies.old
```

To copy a directory, first to a new, and then to an existing destination directory:

```
example% ls ˜/bkup
/usr/example/fred/bkup not found
example% cp –r ˜/src ˜/bkup
example% ls –R ˜/bkup
x.c y.c z.sh
example% cp –r ˜/src ˜/bkup
example% ls –R ˜/bkup
src x.c y.c z.sh

src:
x.c y.c z.sh
```

To copy a list of files to a destination directory:

```
example% cp ˜/src/∗  /tmp
```

SEE ALSO

chmod(1), **cpio**(1), **rm**(1)

NOTES

The permission modes of the source file are preserved in the copy.

A –– permits the user to mark the end of any command line options explicitly, thus allowing **cp** to recognize filename arguments that begin with a –. If a –– and a – both appear on the same command line, the second will be interpreted as a filename.

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio –i [**bBcdfkmrsStuvV6**] [**–C** *bufsize*] [**–E** *filename*] [**–H** *header*]
[**–I** *filename* [**–M** *message*]] [**–R** *id*] [*pattern* . . .]
cpio –o [**aABcLvV**] [**–C** *bufsize*] [**–H** *header*] [**–O** *filename* [**–M** *message*]]
cpio –p [**adlLmuvV**] [**–R** *id*] *directory*

AVAILABILITY

SUNWcsu

DESCRIPTION

cpio copies files in to and out from a **cpio** archive. The **cpio** achive may span multiple volumes. The **–i**, **–o**, and **–p** options select the action to be performed. The following list describes each of the actions (which are mutually exclusive).

cpio –i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio –o**. Only files with names that match *patterns* are selected. *patterns* are regular expressions given in the filename-generating notation of **sh**(1). In *patterns*, meta-characters **?**, ∗, and **[. . .]** match the slash (/) character, and backslash (\) is an escape character. A **!** meta-character means *not*. (For example, the **!abc**∗ pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is ∗ (that is, select all files). Each pattern must be enclosed in double quotes; otherwise, the name of a file in the current directory might be used. Extracted files are conditionally created and copied into the current directory tree based on the options described below. The permissions of the files will be those of the previous **cpio –o**. Owner and group will be the same as the current user unless the current user is super-user. If this is true, owner and group will be the same as those resulting from the previous **cpio –o**. Note: If **cpio –i** tries to create a file that already exists and the existing file is the same age or younger (**newer**), **cpio** will output a warning message and not replace the file. (The **–u** option can be used to overwrite, unconditionally, the existing file.)

cpio –o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary by default or to the user specified block size (with the **–B** or **–C** options) or to some device-dependent block size where necessary (as with the CTC tape).

cpio –p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based on the options described below.

Note: **cpio** assumes four-byte words.

If, when writing to a character device (**–o**) or reading from a character device (**–i**), **cpio** reaches the end of a medium (such as the end of a diskette), and the **–O** and **–I** options are not used, **cpio** prints the following message:

To continue, type device/file name when ready.

To continue, you must replace the medium and type the character special device name (**/dev/rdiskette** for example) and press RETURN. You may want to continue by directing **cpio** to use a different device. For example, if you have two floppy drives you may want to switch between them so **cpio** can proceed while you are changing the floppies. (Simply pressing RETURN causes the **cpio** process to exit.)

OPTIONS

–**i**	**cpio** –**i** (copy in) extracts files from the standard input.
–o	**cpio** –**o** (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output.
–**p**	**cpio** –**p** (pass) reads the standard input to obtain a list of path names of files.
–**a**	Reset access times of input files after they have been copied. Access times are not reset for linked files when **cpio** –**pla** is specified (mutually exclusive with –**m**).
–**A**	Append files to an archive. The –**A** option requires the –**O** option. Valid only with archives that are files, or that are on floppy diskettes or hard disk partitions.
–**b**	Reverse the order of the bytes within each word. (Use only with the –**i** option.)
–**B**	Block input/output 5120 bytes to the record. The default buffer size is 512 bytes when this and the –**C** options are not used. (–**B** does not apply to the *pass* option; –**B** is meaningful only with data directed to or from a character special device, for example, **/dev/rmt/0m .)**
–**c**	Read or write header information in ASCII character form for portability. Always use this option (or the –**H** option) when the origin and the destination machines are different types (mutually exclusive with –**H** and –**6**). The –**c** option implies expanded device numbers. When transferring files between Solaris 1.x and Solaris 2.x use regular **cpio** format. Don't use the –**c** or –**H** options.
–**C** *bufsize*	Block input/output *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this and –**B** options are not used. (–**C** does not apply to the *pass* option; –**C** is meaningful only with data directed to or from a character special device, for example, **/dev/rmt/0m**.)
–**d**	Create directories as needed.
–**E** *filename*	Specify an input file (*file*) that contains a list of filenames to be extracted from the archive (one filename per line).
–**f**	Copy in all files except those in *patterns.* (See the paragraph on **cpio** –**i** for a description of *patterns*.)
–**H** *header*	Read or write header information in *header* format. Always use this option or the –**c** option when the origin and the destination machines are different types (mutually exclusive with –**c** and –**6**). When

transferring files between Solaris 1.x and Solaris 2.x use regular **cpio** format. Don't use the –**c** or –**H** options.

Valid values for *header* are:

bar	**bar** head and format. Used only with the –**i** option (read only)
crc | **CRC**	ASCII header with expanded device numbers and an additional per-file checksum
odc	ASCII header with small device numbers
tar | **TAR**	**tar** header and format
ustar | **USTAR**	IEEE / P1003 Data Interchange Standard header and format

–**I** *filename*	Read the contents of *filename* as an input archive. If *filename* is a character special device, and the current medium has been completely read, replace the medium and press RETURN to continue to the next medium. This option is used only with the –**i** option.
–**k**	Attempt to skip corrupted file headers and I / O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For **cpio** archives that contain other **cpio** archives, if an error is encountered **cpio** may terminate prematurely. **cpio** will find the next good header, which may be one for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the –**i** option.
–**l**	Whenever possible, link files rather than copying them. (Usable only with the –**p** option.)
–**L**	Follow symbolic links. The default is not to follow symbolic links.
–**m**	Retain previous file modification time. This option is ineffective on directories that are being copied (mutually exclusive with –**a**).
–**M** *message*	Define a *message* to use when switching media. When you use the –**O** or –**I** options and specify a character special device, you can use this option to define the message that is printed when you reach the end of the medium. One **%d** can be placed in *message* to print the sequence number of the next medium needed to continue.
–**O** *filename*	Direct the output of **cpio** to *filename*. If *filename* is a character special device and the current medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the –**o** option.
–**r**	Interactively rename files. If the user types a carriage return alone, the file is skipped. If the user types a ‘‘.’’ the original pathname will be retained. (Not available with **cpio** –**p**.)
–**R** *id*	Reassign ownership and group information for each file to *user ID* (*ID*

	must be a valid login ID from **/etc/passwd**). This option is valid only for the super-user.
−s	Swap bytes within each half word.
−S	Swap halfwords within each word.
−t	Print a table of contents of the input. No files are created (mutually exclusive with −**V**).
−u	Copy unconditionally (normally, an older file will not replace a newer file with the same name).
−v	Verbose. Print a list of file names. When used with the −**t** option, the table of contents looks like the output of an **ls** −**l** command (see **ls**(1)).
−V	Special verbose. Print a dot for each file read or written. Useful to assure the user that **cpio** is working without printing out all file names.
−6	Process a UNIX System Sixth Edition archive format file. Use only with the −**i** option (mutually exclusive with −**c** and −**H**)).

EXAMPLES

The following examples show three uses of **cpio**.

When standard input is directed through a pipe to **cpio** −**o**, it groups the files so they can be directed (>) to a single file (**../newfile**). The −**c** option insures that the file will be portable to other machines (as would the −**H** option). Instead of **ls**(1), you could use **find**(1), **echo**(1), **cat**(1), and so on, to pipe a list of names to **cpio**. You could direct the output to a device instead of a file.

```
example% ls |  cpio  −oc > ../newfile
```

cpio −**i** uses the output file of **cpio** −**o** (directed through a pipe with **cat** in the example below), extracts those files that match the patterns (**memo/a1**, **memo/b∗**), creates directories below the current directory as needed (−**d** option), and places the files in the appropriate directories. The −**c** option is used if the input file was created with a portable header. If no patterns were given, all files from **newfile** would be placed in the directory.

```
example% cat newfile |  cpio  −icd "memo/a1" "memo/b∗"
```

cpio −**p** takes the file names piped to it and copies or links (−**l** option) those files to another directory (**newdir** in the example below). The −**d** option says to create directories as needed. The −**m** option says retain the modification time. (It is important to use the −**depth** option of **find**(1) to generate path names for **cpio**. This eliminates problems **cpio** could have trying to create files under read-only directories.) The destination directory, **newdir**, must exist.

```
example% find . −depth −print |  cpio  −pdlmv  newdir
```

Note: When you use **cpio** in conjunction with **find**, if you use the −**L** option with **cpio** then you must use the −**follow** option with **find** and vice versa. Otherwise there will be undesirable results.

Note that for multi-reel archives, dismount the old volume, mount the new one, and continue to the next tape by typing the name of the next device (probably the same as the first reel). To stop, type a RETURN and **cpio** will end.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **cpio** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **cpio** behaves.

LC_CTYPE

Determines how **cpio** handles characters. When **LC_CTYPE** is set to a valid value, **cpio** can display and handle text and filenames containing valid characters for that locale. **cpio** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **cpio** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME

Determines how **cpio** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

SEE ALSO

ar(1), **cat**(1), **echo**(1), **find**(1), **ls**(1), **sh**(1), **tar**(1), **archives**(4), **environ**(5)

NOTES

Path names are restricted to 256 characters for the binary (the default) and **−H odc** header formats. Otherwise, path names are restricted to 1024 characters.

Only the super-user can copy special files.

Blocks are reported in 512-byte quantities.

If a file has **000** permissions, contains more than 0 characters of data, and the user is not root, the file will not be saved or restored.

The inode number stored in the header, (**/usr/include/archives.h**) is an unsigned short which is 2 bytes. This limits the range of inode numbers from 0 to 65535. Files which are hard linked must fall in this inode range. This could be a problem when moving **cpio** archives between different vendors’ machines.

NAME

cpp – the C language preprocessor

SYNOPSIS

/usr/ccs/lib/cpp [–**BCHMpPRT**] [–**undef**] [–**D***name*] [–**D***name***=***def*] [–**I***directory*] [–**U***name*] [–**Y***directory*] [*input-file* [*output-file*]]

DESCRIPTION

cpp is the C language preprocessor. It is invoked as the first pass of any C compilation started with the **cc**(1B) command; however, **cpp** can also be used as a first-pass preprocessor for other Sun compilers.

Although **cpp** can be used as a macro processor, this is not normally recommended, as its output is geared toward that which would be acceptable as input to a compiler's second pass. Thus, the preferred way to invoke **cpp** is through the **cc**(1B) command, or some other compilation command. For general-purpose macro-processing, see **m4**(1), and the chapter on **m4** in *Programming Utilities Guide*.

cpp optionally accepts two filenames as arguments. *input-file* and *output-file* are, respectively, the input and output files for the preprocessor. They default to the standard input and the standard output.

OPTIONS

–**B**	Support the C++ comment indicator '**//**'. With this indicator everything on the line after the **//** is treated as a comment.
–**C**	Pass all comments (except those that appear on **cpp** directive lines) through the preprocessor. By default, **cpp** strips out C-style comments.
–**H**	Print the pathnames of included files, one per line on the standard error.
–**M**	Generate a list of makefile dependencies and write them to the standard output. This list indicates that the object file which would be generated from the input file depends on the input file as well as the include files referenced.
–**p**	Use only the first eight characters to distinguish preprocessor symbols, and issue a warning if extra tokens appear at the end of a line containing a directive.
–**P**	Preprocess the input without producing the line control information used by the next pass of the C compiler.
–**R**	Allow recursive macros.
–**T**	Use only the first eight characters for distinguishing different preprocessor names. This option is included for backward compatibility with systems which always use only the first eight characters.
–**undef**	Remove initial definitions for all predefined symbols.
–**D***name*	Define *name* as 1 (one). This is the same as if a –**D***name***=1** option appeared on the **cpp** command line, or as if a **#define** *name* **1** line appeared in the source file that **cpp** is processing.

–**D***name***=***def*

Define *name* as if by a **#define** directive. This is the same as if a

#define *name* **def**

line appeared in the source file that **cpp** is processing. The –**D** option has lower precedence than the –**U** option. That is, if the same name is used in both a –**U** option and a –**D** option, the name will be undefined regardless of the order of the options.

–**I***directory*

Insert *directory* into the search path for **#include** files with names not beginning with '/'. *directory* is inserted ahead of the standard list of ''include'' directories. Thus, **#include** files with names enclosed in double-quotes (") are searched for first in the directory of the file with the **#include** line, then in directories named with –**I** options, and lastly, in directories from the standard list. For **#include** files with names enclosed in angle-brackets (**< >**), the directory of the file with the **#include** line is not searched. See **Details** below for exact details of this search order.

–**U***name*

Remove any initial definition of *name*, where *name* is a symbol that is predefined by a particular preprocessor. Here is a partial list of symbols that may be predefined, depending upon the architecture of the system:

Operating System: **ibm**, **gcos**, **os**, **tss** and **unix**
Hardware: **interdata**, **pdp11**, **u370**, **u3b**, **u3b2**, **u3b5**, **u3b15**, **u3b20d**, **vax**, **ns32000**, **iAPX286**, **i386**, **sparc**, and **sun**
UNIX system variant: **RES**, and **RT**
The **lint** command: **lint**

The symbols **sun**, **sparc** and **unix** are defined for all Sun systems.

–**Y***directory*

Use directory *directory* in place of the standard list of directories when searching for **#include** files.

USAGE

Directives

All **cpp** directives start with a hash symbol (**#**) as the first character on a line. White space (SPACE or TAB characters) can appear after the initial **#** for proper indentation.

#define *name token-string*

Replace subsequent instances of *name* with *token-string*.

#define *name***(***argument* [**,** *argument*] . . . **)** *token-string*

There can be no space between *name* and the '**(**'. Replace subsequent instances of *name*, followed by a parenthesized list of arguments, with *token-string*, where each occurrence of an *argument* in the *token-string* is replaced by the corresponding token in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#undef *name*

Remove any definition for the symbol *name*. No additional tokens are permitted on the directive line after *name*.

#include "*filename*"
#include <*filename*>

Read in the contents of *filename* at this location. This data is processed by **cpp** as if it were part of the current file. When the <*filename*> notation is used, *filename* is only searched for in the standard ‘‘include’’ directories. See the –**I** and –**Y** options above for more detail. No additional tokens are permitted on the directive line after the final ‘"’ or ‘>’.

#line *integer-constant* "*filename*"

Generate line control information for the next pass of the C compiler. *integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file from where it comes. If "*filename*" is not given, the current filename is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

#if *constant-expression*

Subsequent lines up to the matching **#else**, **#elif ,** or **#endif** directive, appear in the output only if *constant-expression* yields a nonzero value. All binary non-assignment C operators, including ‘**&&**’, ‘**| |**’, and ‘**,**’, are legal in *constant-expression*. The ‘**?:**’ operator, and the unary ‘–’, ‘**!**’, and ‘**~**’ operators, are also legal in *constant-expression*.

The precedence of these operators is the same as that for C. In addition, the unary operator **defined**, can be used in *constant-expression* in these two forms: ‘**defined (** *name* **)**’ or ‘**defined** *name*’. This allows the effect of **#ifdef** and **#ifndef** directives (described below) in the **#if** directive. Only these operators, integer constants, and names that are known by **cpp** should be used within *constant-expression*. In particular, the **size of** operator is not available.

#ifdef *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has been defined, either with a **#define** directive or a –**D** option, and in the absence of an intervening **#undef** directive. Additional tokens after *name* on the directive line will be silently ignored.

#ifndef *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has *not* been defined, or if its definition has been removed with an **#undef** directive. No additional tokens are permitted on the directive line after *name* .

#elif *constant-expression*

Any number of **#elif** directives may appear between an **#if**, **#ifdef**, or **#ifndef** directive and a matching **#else** or **#endif** directive. The lines following the **#elif** directive appear in the output only if all of the following conditions hold:

- The *constant-expression* in the preceding **#if** directive evaluated to zero, the *name* in the preceding **#ifdef** is not defined, or the *name* in the preceding **#ifndef** directive *was* defined.

- The *constant-expression* in all intervening **#elif** directives evaluated to zero.
- The current *constant-expression* evaluates to non-zero.

If the *constant-expression* evaluates to non-zero, subsequent **#elif** and **#else** directives are ignored up to the matching **#endif**. Any *constant-expression* allowed in an **#if** directive is allowed in an **#elif** directive.

#else This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the **#else** and the matching **#endif** are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included in the output. Conditional directives and corresponding **#else** directives can be nested.

#endif End a section of lines begun by one of the conditional directives **#if**, **#ifdef**, or **#ifndef**. Each such directive must have a matching **#endif**.

Macros

Formal parameters for macros are recognized in **#define** directive bodies, even when they occur inside character constants and quoted strings. For instance, the output from:

```
#define abc(a)  |`|a|
abc(xyz)
```

is the seven characters ‘‘ **|`xyz|**’’ (SPACE, vertical-bar, backquote, x, y, z, vertical-bar). Macro names are not recognized within character constants or quoted strings during the regular scan. Thus:

```
#define abc  xyz
printf("abc");
```

does not expand **abc** in the second line, since it is inside a quoted string that is not part of a **#define** macro definition.

Macros are not expanded while processing a **#define** or **#undef**. Thus:

```
#define abc zingo
#define xyz abc
#undef abc
xyz
```

produces **abc**. The token appearing immediately after an **#ifdef** or **#ifndef** is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces “ **#define hello goodbye hello**”.

Output

Output consists of a copy of the input file, with modifications, plus lines of the form:

#lineno " *filename* " "*level* "

indicating the original source line number and filename of the following output line and whether this is the first such line after an include file has been entered (*level*=**1**), the first such line after an include file has been exited (*level*=**2**), or any other such line (*level* is empty).

Details

Directory Search Order

#include files are searched for in the following order:

1. The directory of the file that contains the **#include** request (that is, **#include** is relative to the file being scanned when the request is made).
2. The directories specified by –**I** options, in left-to-right order.
3. The standard directory(s) (**/usr/include** on UNIX systems).

Special Names

Two special names are understood by **cpp**. The name **_ _LINE_ _** is defined as the current line number (a decimal integer) as known by **cpp**, and **_ _FILE_ _** is defined as the current filename (a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

Newline Characters

A NEWLINE character terminates a character constant or quoted string. An escaped NEWLINE (that is, a backslash immediately followed by a NEWLINE) may be used in the body of a **#define** statement to continue the definition onto the next line. The escaped NEWLINE is not included in the macro value.

Comments

Comments are removed (unless the –**C** option is used on the command line). Comments are also ignored, except that a comment terminates a token.

SEE ALSO

cc(1B), **m4**(1)

Programming Utilities Guide

DIAGNOSTICS

The error messages produced by **cpp** are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When NEWLINE characters were found in argument lists for macros to be expanded, some previous versions of **cpp** put out the NEWLINE characters as they were found and expanded. The current version of **cpp** replaces them with SPACE characters.

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

#include <file.h>

should be used, rather than one with an absolute path, like:

#include "/usr/include/file.h"

cpp warns about the use of the absolute pathname.

While the compiler allows 8-bit strings and comments, 8-bits are not allowed anywhere else.

NAME

crontab – user crontab file

SYNOPSIS

crontab [*filename*]
crontab –e [*username*]
crontab –r [*username*]
crontab –l [*username*]

AVAILABILITY

SUNWcsu

DESCRIPTION

crontab manages a user's access with **cron** by copying, creating, listing, and removing **crontab** files. If invoked without options, **crontab** copies the specified file, or the standard input if no file is specified, into a directory that holds all users' crontabs.

crontab Access Control

Users: Access to **crontab** is allowed:

- if the user's name appears in **/etc/cron.d/cron.allow**.
- if **/etc/cron.d/cron.allow** does not exist and the user's name is not in **/etc/cron.d/cron.deny**.

Users: Access to **crontab** is denied:

- if **/etc/cron.d/cron.allow** exists and the user's name is not in it.
- if **/etc/cron.d/cron.allow** does not exist and user's name is in **/etc/cron.d/cron.deny**.
- if neither file exists.

Note: The rules for allow and deny apply to **root** only if the allow / deny files exist.

The allow / deny files consist of one user name per line.

crontab Entry Format

A **crontab** file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (**0**–**59**),
hour (**0**–**23**),
day of the month (**1**–**31**),
month of the year (**1**–**12**),
day of the week (**0**–**6** with **0**=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, **0 0 1,15** ∗ **1** would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to ∗ (for example, **0 0** ∗ ∗ **1** would run a command only on Mondays).

The sixth field of a line in a **crontab** file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a **%** or end of line) of the command field is executed by the shell. Other lines are made available to the command as standard input. Any line beginning with a **#** is a comment and will be ignored.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh.** Users who desire to have their **.profile** executed must explicitly do so in the **crontab** file. **cron** supplies a default environment for every shell, defining **HOME**, **LOGNAME**, **SHELL(=/bin/sh)**, **TZ**, and **PATH**. The default **PATH** for **user** cron jobs is **/usr/bin**; while **root** cron jobs default to **/usr/sbin:/usr/bin**. The default **PATH** can be set in **/etc/default/cron**. (See **cron**(1M)).

If you do not redirect the standard output and standard error of your commands, any generated output or errors will be mailed to you.

OPTIONS

–e edits a copy of the current user's **crontab** file, or creates an empty file to edit if **crontab** does not exist. When editing is complete, the file is installed as the user's **crontab** file. If a *username* is given, the specified user's **crontab** file is edited, rather than the current user's **crontab** file; this may only be done by a privileged user. The environment variable **EDITOR** determines which editor is invoked with the **–e** option. The default editor is **ed**(1).

Note: All crontab jobs should be submitted using **crontab**; you should not add jobs by just editing **crontab** file because **cron** will not be aware of changes made this way.

–l lists the **crontab** file for the invoking user. Only a privileged user can specify a *username* following the **–r** or **–l** options to remove or list the **crontab** file of the specified user.

–r removes a user's crontab from the crontab directory.

FILES

/etc/cron.d	main cron directory
/etc/cron.d/cron.allow	list of allowed users
/etc/default/cron	contains cron default settings
/etc/cron.d/cron.deny	list of denied users
/var/cron/log	accounting information
/var/spool/cron/crontabs	spool area for **crontab**.

SEE ALSO

atq(1), **atrm**(1), **ed**(1), **sh**(1), **cron**(1M), **su**(1M)

NOTES

If you inadvertently enter the **crontab** command with no argument(s), do not attempt to get out with a CTRL-D. This removes all entries in your **crontab** file. Instead, exit with a CTRL-C.

If a privileged user modifies another user's **crontab** file, resulting behavior may be unpredictable. Instead, the privileged user should first **su**(1M) to the other user's login before making any changes to the **crontab** file.

NAME

crypt – encode or decode a file

SYNOPSIS

crypt [*password*]

AVAILABILITY

SUNWcry

DESCRIPTION

crypt encrypts and decrypts the contents of a file. **crypt** reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, **crypt** demands a key from the terminal and turns off printing while the key is being typed in. **crypt** encrypts and decrypts with the same key:

```
example% crypt key <clear.file>encrypted.file
example% crypt key <encrypted.file |  pr
```

will print the contents of *clear.file*.

Files encrypted by **crypt** are compatible with those treated by the editors **ed**(1), **ex**(1) and **vi**(1) in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; “sneak paths” by which keys or cleartext can become visible must be minimized.

crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are widely known, thus **crypt** provides minimal security.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, that is, to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the **crypt** command, it is potentially visible to users executing **ps**(1) or a derivative command. To minimize this possibility, **crypt** takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of **crypt.**

FILES

/dev/tty	for typed key

SEE ALSO

des(1), **ed**(1), **ex**(1), **ps**(1), **vi**(1), **makekey**(1)

RESTRICTIONS

This program is not available on software shipped outside the U.S.

NAME

csh – shell command interpreter with a C-like syntax

SYNOPSIS

csh [**–bcefinstvVxX**] [*argument*...]

AVAILABILITY

SUNWcsu

DESCRIPTION

csh, the C shell, is a command interpreter with a syntax reminiscent of the C language. It provides a number of convenient features for interactive use that are not available with the Bourne shell, including filename completion, command aliasing, history substitution, job control, and a number of built-in commands. As with the Bourne shell, the C shell provides variable, command and filename substitution.

Initialization and Termination

When first started, the C shell normally performs commands from the **.cshrc** file in your home directory, provided that it is readable and you either own it or your real group ID matches its group ID. If the shell is invoked with a name that starts with '–', as when started by **login**(1), the shell runs as a **login** shell. In this case, after executing commands from the **.cshrc** file, the shell executes commands from the **.login** file in your home directory; the same permission checks as those for **.cshrc** are applied to this file. Typically, the **.login** file contains commands to specify the terminal type and environment.

As a login shell terminates, it performs commands from the **.logout** file in your home directory; the same permission checks as those for **.cshrc** are applied to this file.

Interactive Operation

After startup processing is complete, an interactive C shell begins reading commands from the terminal, prompting with *hostname***%** (or *hostname***#** for the privileged user). The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the history list and then parsed, as described under USAGE, below. Finally, the shell executes each command in the current line.

Noninteractive Operation

When running noninteractively, the shell does not prompt for input from the terminal. A noninteractive C shell can execute a command supplied as an *argument* on its command line, or interpret commands from a file, also known as a script.

OPTIONS

–b Force a "break" from option processing. Subsequent command line arguments are not interpreted as C shell options. This allows the passing of options to a script without confusion. The shell does not run set-user-ID or set-group-ID scripts unless this option is present.

–c Read commands from the first filename *argument* (which must be present). Remaining arguments are placed in **argv**, the argument-list variable, and passed directly to **csh**.

–e Exit if a command terminates abnormally or yields a nonzero exit status.

–f Fast start. Read neither the **.cshrc** file, nor the **.login** file (if a login shell) upon startup.

–i	Forced interactive. Prompt for command line input, even if the standard input does not appear to be a terminal (character-special device).
–n	Parse (interpret), but do not execute commands. This option can be used to check C shell scripts for syntax errors.
–s	Take commands from the standard input.
–t	Read and execute a single command line. A '\' (backslash) can be used to escape each newline for continuation of the command line onto subsequent input lines.
–v	Verbose. Set the **verbose** predefined variable; command input is echoed after history substitution (but before other substitutions) and before execution.
–V	Set **verbose** before reading **.cshrc**.
–x	Echo. Set the **echo** variable; echo commands after all substitutions and just before execution.
–X	Set **echo** before reading **.cshrc**.

Except with the options –**c**, –**i**, –**s**, or –**t**, the first nonoption *argument* is taken to be the name of a command or script. It is passed as argument zero, and subsequent arguments are added to the argument list for that command or script.

USAGE

Filename Completion

When enabled by setting the variable **filec**, an interactive C shell can complete a partially typed filename or user name. When an unambiguous partial filename is followed by an ESC character on the terminal input line, the shell fills in the remaining characters of a matching filename from the working directory.

If a partial filename is followed by the EOF character (usually typed as CTRL-d), the shell lists all filenames that match. It then prompts once again, supplying the incomplete command line typed in so far.

When the last (partial) word begins with a tilde (~), the shell attempts completion with a user name, rather than a file in the working directory.

The terminal bell signals errors or multiple matches; this can be inhibited by setting the variable **nobeep**. You can exclude files with certain suffixes by listing those suffixes in the variable **fignore**. If, however, the only possible completion includes a suffix in the list, it is not ignored. **fignore** does not affect the listing of filenames by the EOF character.

Lexical Structure

The shell splits input lines into words at space and tab characters, except as noted below. The characters **&**, **|**, **;**, **<**, **>**, **(**, and **)** form separate words; if paired, the pairs form single words. These shell metacharacters can be made part of other words, and their special meaning can be suppressed by preceding them with a '\' (backslash). A newline preceded by a \ is equivalent to a space character.

In addition, a string enclosed in matched pairs of single-quotes (´), double-quotes ("), or backquotes (`), forms a partial word; metacharacters in such a string, including any space or tab characters, do not form separate words. Within pairs of backquote (`**)** or double-quote (") characters, a newline preceded by a '\' (backslash) gives a true newline character. Additional functions of each type of quote are described, below, under **Variable Substitution**, **Command Substitution**, and **Filename Substitution**.

When the shell's input is not a terminal, the character **#** introduces a comment that continues to the end of the input line. Its special meaning is suppressed when preceded by a \ or enclosed in matching quotes.

Command Line Parsing

A *simple command* is composed of a sequence of words. The first word (that is not part of an I/O redirection) specifies the command to be executed. A simple command, or a set of simple commands separated by **|** or **|&** characters, forms a *pipeline*. With **|**, the standard output of the preceding command is redirected to the standard input of the command that follows. With **| &**, both the standard error and the standard output are redirected through the pipeline.

Pipelines can be separated by semicolons (**;**), in which case they are executed sequentially. Pipelines that are separated by **&&** or **| |** form conditional sequences in which the execution of pipelines on the right depends upon the success or failure, respectively, of the pipeline on the left.

A pipeline or sequence can be enclosed within parentheses '**( )**' to form a simple command that can be a component in a pipeline or sequence.

A sequence of pipelines can be executed asynchronously or "in the background" by appending an '**&**'; rather than waiting for the sequence to finish before issuing a prompt, the shell displays the job number (see **Job Control**, below) and associated process IDs and prompts immediately.

History Substitution

History substitution allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments. Command lines are saved in the history list, the size of which is controlled by the **history** variable. The most recent command is retained in any case. A history substitution begins with a **!** (although you can change this with the **histchars** variable) and may occur anywhere on the command line; history substitutions do not nest. The **!** can be escaped with \ to suppress its special meaning.

Input lines containing history substitutions are echoed on the terminal after being expanded, but before any other substitutions take place or the command gets executed.

Event Designators

An event designator is a reference to a command line entry in the history list.

!	Start a history substitution, except when followed by a space character, tab, newline, **=** or **(**.
!!	Refer to the previous command. By itself, this substitution repeats the previous command.
!*n*	Refer to command line *n*.
!–*n*	Refer to the current command line minus *n*.

!_str_	Refer to the most recent command starting with _str_.
!?_str_[**?**]	Refer to the most recent command containing _str_.
!{. . .**}**	Insulate a history reference from adjacent characters (if necessary).

Word Designators

A ‘**:**’ (colon) separates the event specification from the word designator. It can be omitted if the word designator begins with a **ˆ**, **$**, ∗, – or **%**. If the word is to be selected from the previous command, the second **!** character can be omitted from the event specification. For instance, **!!:1** and **!:1** both refer to the first word of the previous command, while **!!$** and **!$** both refer to the last word in the previous command. Word designators include:

#	The entire command line typed so far.
0	The first input word (command).
_n_	The _n_’th argument.
ˆ	The first argument, that is, **1**.
$	The last argument.
%	The word matched by (the most recent) **?**_s_ search.
_x_–_y_	A range of words; –_y_ abbreviates **0**–_y_.
∗	All the arguments, or a null value if there is just one word in the event.
_x_**∗**	Abbreviates _x_–**$**.
_x_–	Like _x_∗ but omitting word **$**.

Modifiers

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a **:**.

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing suffix of the form ‘**.**_xxx_’, leaving the basename.
e	Remove all but the suffix.
s/_l_**/**_r_[**/**]	Substitute _r_ for _l_.
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change to the first occurrence of a match in each word, by prefixing the above (for example, **g&**).
p	Print the new command but do not execute it.
q	Quote the substituted words, escaping further substitutions.
x	Like **q**, but break into words at each space character, tab or newline.

Unless preceded by a **g**, the modification is applied only to the first string that matches _l_; an error results if no string matches.

The left-hand side of substitutions are not regular expressions, but character strings. Any character can be used as the delimiter in place of **/**. A backslash quotes the delimiter character. The character **&**, in the right hand side, is replaced by the text from the left-hand-side. The **&** can be quoted with a backslash. A null _l_ uses the previous string either from a _l_ or from a contextual scan string _s_ from **!?**_s_. You can omit the rightmost delimiter if a newline immediately follows _r_; the rightmost **?** in a context scan can similarly be omitted.

Without an event specification, a history reference refers either to the previous command, or to a previous history reference on the command line (if any).

Quick Substitution

^*l*^*r*[^] This is equivalent to the history substitution: **!:s/***l***/***r*[**/**].

Aliases

The C shell maintains a list of aliases that you can create, display, and modify using the **alias** and **unalias** commands. The shell checks the first word in each command to see if it matches the name of an existing alias. If it does, the command is reprocessed with the alias definition replacing its name; the history substitution mechanism is made available as though that command were the previous input line. This allows history substitutions, escaped with a backslash in the definition, to be replaced with actual command line arguments when the alias is used. If no history substitution is called for, the arguments remain unchanged.

Aliases can be nested. That is, an alias definition can contain the name of another alias. Nested aliases are expanded before any history substitutions is applied. This is useful in pipelines such as

```
alias lm ´ls –l \!* | more´
```

which when called, pipes the output of **ls**(1) through **more**(1).

Except for the first word, the name of the alias may not appear in its definition, nor in any alias referred to by its definition. Such loops are detected, and cause an error message.

I/O Redirection

The following metacharacters indicate that the subsequent word is the name of a file to which the command’s standard input, standard output, or standard error is redirected; this word is variable, command, and filename expanded separately from the rest of the command.

<	Redirect the standard input.
<< *word*	Read the standard input, up to a line that is identical with *word*, and place the resulting lines in a temporary file. Unless *word* is escaped or quoted, variable and command substitutions are performed on these lines. Then, the pipeline is invoked with the temporary file as its standard input. *word* is not subjected to variable, filename, or command substitution, and each line is compared to it before any substitutions are performed by the shell.
> >! >& >&!	Redirect the standard output to a file. If the file does not exist, it is created. If it does exist, it is overwritten; its previous contents are lost. When set, the variable **noclobber** prevents destruction of existing files. It also prevents redirection to terminals and **/dev/null**, unless one of the **!** forms is used. The **&** forms redirect both standard output and the standard error (diagnostic output) to the file.

>> >>& >>! >>&!
Append the standard output. Like **>**, but places output at the end of the file rather than overwriting it. If **noclobber** is set, it is an error for the file not to exist, unless one of the **!** forms is used. The **&** forms append both the standard error and standard output to the file.

Variable Substitution

The C shell maintains a set of *variables*, each of which is composed of a *name* and a *value*. A variable name consists of up to 20 letters and digits, and starts with a letter (the underscore is considered a letter). A variable's value is a space-separated list of zero or more words.

To refer to a variable's value, precede its name with a '**$**'. Certain references (described below) can be used to select specific words from the value, or to display other information about the variable. Braces can be used to insulate the reference from other characters in an input-line word.

Variable substitution takes place after the input line is analyzed, aliases are resolved, and I/O redirections are applied. Exceptions to this are variable references in I/O redirections (substituted at the time the redirection is made), and backquoted strings (see Command Substitution).

Variable substitution can be suppressed by preceding the **$** with a \, except within double-quotes where it always occurs. Variable substitution is suppressed inside of single-quotes. A **$** is escaped if followed by a space character, tab or newline.

Variables can be created, displayed, or destroyed using the **set** and **unset** commands. Some variables are maintained or used by the shell. For instance, the **argv** variable contains an image of the shell's argument list. Of the variables used by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not.

Numerical values can be operated on as numbers (as with the **@** built-in command). With numeric operations, an empty value is considered to be zero; the second and subsequent words of multiword values are ignored. For instance, when the **verbose** variable is set to any value (including an empty value), command input is echoed on the terminal.

Command and filename substitution is subsequently applied to the words that result from the variable substitution, except when suppressed by double-quotes, when **noglob** is set (suppressing filename substitution), or when the reference is quoted with the **:q** modifier. Within double-quotes, a reference is expanded to form (a portion of) a quoted string; multiword values are expanded to a string with embedded space characters. When the **:q** modifier is applied to the reference, it is expanded to a list of space-separated words, each of which is quoted to prevent subsequent command or filename substitutions.

Except as noted below, it is an error to refer to a variable that is not set.

$*var*
${*var***}**
These are replaced by words from the value of *var*, each separated by a space character. If *var* is an environment variable, its value is returned (but '**:**' modifiers and the other forms given below are not available).

$***var*[*index*] **${*var***[***index***]}**	These select only the indicated words from the value of *var*. Variable substitution is applied to *index* , which may consist of (or result in) a either single number, two numbers separated by a '–', or an asterisk. Words are indexed starting from 1; a '*' selects all words. If the first number of a range is omitted (as with **$argv[–2]**), it defaults to 1. If the last number of a range is omitted (as with **$argv[1–]**), it defaults to **$#***var* (the word count). It is not an error for a range to be empty if the second argument is omitted (or within range).
$#*name* **${#***name***}**	These give the number of words in the variable.
$0	This substitutes the name of the file from which command input is being read. An error occurs if the name is not known.
$*n* **${***n***}**	Equivalent to **$argv[***n***]**.
$***	Equivalent to **$argv[*]**.

The modifiers **:e**, **:h**, **:q**, **:r**, **:t**, and **:x** can be applied (see **History Substitution**), as can **:gh**, **:gt**, and **:gr**. If **{ }** (braces) are used, then the modifiers must appear within the braces. The current implementation allows only one such modifier per expansion.

The following references may not be modified with **:** modifiers.

$?*var* **${?***var***}**	Substitutes the string 1 if *var* is set or 0 if it is not set.
$?0	Substitutes 1 if the current input filename is known or 0 if it is not.
$$	Substitute the process number of the (parent) shell.
$<	Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a C shell script.

Command and Filename Substitutions

Command and filename substitutions are applied selectively to the arguments of built-in commands. Portions of expressions that are not evaluated are not expanded. For non-built-in commands, filename expansion of the command name is done separately from that of the argument list; expansion occurs in a subshell, after I/O redirection is performed.

Command Substitution

A command enclosed by backquotes (`...`) is performed by a subshell. Its standard output is broken into separate words at each space character, tab and newline; null words are discarded. This text replaces the backquoted string on the current command line. Within double-quotes, only newline characters force new words; space and tab characters are preserved. However, a final newline is ignored. It is therefore possible for a command substitution to yield a partial word.

Filename Substitution

Unquoted words containing any of the characters *, **?**, **[** or **{**, or that begin with ~, are expanded (also known as *globbing*) to an alphabetically sorted list of filenames, as follows:

*	Match any (zero or more) characters.
?	Match any single character.
[. . . **]**	Match any single character in the enclosed list(s) or range(s). A list is a string of characters. A range is two characters separated by a dash (–), and includes all the characters in between in the ASCII collating sequence (see **ascii**(5)).
{ *str***,** *str***,** . . . **}**	Expand to each string (or filename-matching pattern) in the comma-separated list. Unlike the pattern-matching expressions above, the expansion of this construct is not sorted. For instance, **{b,a}** expands to '**b**' '**a**', (not '**a**' '**b**'). As special cases, the characters **{** and **}**, along with the string **{}**, are passed undisturbed.
~[*user*]	Your home directory, as indicated by the value of the variable **home**, or that of *user*, as indicated by the password entry for *user*.

Only the patterns *, **?** and **[**. . .**]** imply pattern matching; an error results if no filename matches a pattern that contains them. The '**.**' (dot character), when it is the first character in a filename or pathname component, must be matched explicitly. The **/** (slash) must also be matched explicitly.

Expressions and Operators

A number of C shell built-in commands accept expressions, in which the operators are similar to those of C and have the same precedence. These expressions typically appear in the **@**, **exit**, **if**, **set** and **while** commands, and are often used to regulate the flow of control for executing commands. Components of an expression are separated by white space.

Null or missing values are considered 0. The result of all expressions is a string, which may represent decimal numbers.

The following C shell operators are grouped in order of precedence:

(. . . **)**	grouping
~	one's complement
!	logical negation
* **/** **%**	multiplication, division, remainder (These are right associative, which can lead to unexpected results. Group combinations explicitly with parentheses.)
+ –	addition, subtraction (also right associative)
<< **>>**	bitwise shift left, bitwise shift right
< **>** **<=** **>=**	less than, greater than, less than or equal to, greater than or equal to
= = **!=** **=~** **!~**	equal to, not equal to, filename-substitution pattern match (described below), filename-substitution pattern mismatch

&	bitwise AND
^	bitwise XOR (exclusive or)
|	bitwise inclusive OR
&&	logical AND
||	logical OR

The operators: **==**, **!=**, **=~**, and **!~** compare their arguments as strings; other operators use numbers. The operators **=~** and **!~** each check whether or not a string to the left matches a filename substitution pattern on the right. This reduces the need for **switch** statements when pattern-matching between strings is all that is required.

Also available are file inquiries:

–**r** *filename*	Return true, or 1 if the user has read access. Otherwise it returns false, or 0.
–**w** *filename*	True if the user has write access.
–**x** *filename*	True if the user has execute permission (or search permission on a directory).
–**e** *filename*	True if *filename* exists.
–**o** *filename*	True if the user owns *filename*.
–**z** *filename*	True if *filename* is of zero length (empty).
–**f** *filename*	True if *filename* is a plain file.
–**d** *filename*	True if *filename* is a directory.

If *filename* does not exist or is inaccessible, then all inquiries return false.

An inquiry as to the success of a command is also available:

{ *command* **}**	If *command* runs successfully, the expression evaluates to true, 1. Otherwise, it evaluates to false, 0. (Note: Conversely, *command* itself typically returns 0 when it runs successfully, or some other value if it encounters a problem. If you want to get at the status directly, use the value of the **status** variable rather than this expression).

Control Flow

The shell contains a number of commands to regulate the flow of control in scripts and within limits, from the terminal. These commands operate by forcing the shell either to reread input (to *loop*), or to skip input under certain conditions (to *branch*).

Each occurrence of a **foreach**, **switch**, **while**, **if**. . .**then** and **else** built-in command must appear as the first word on its own input line.

If the shell's input is not seekable and a loop is being read, that input is buffered. The shell performs seeks within the internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward **goto** commands will succeed on nonseekable inputs.)

Command Execution

If the command is a C shell built-in command, the shell executes it directly. Otherwise, the shell searches for a file by that name with execute access. If the command name contains a **/**, the shell takes it as a pathname, and searches for it. If the command name does not contain a **/**, the shell attempts to resolve it to a pathname, searching each directory in the **path** variable for the command. To speed the search, the shell uses its hash table (see the **rehash** built-in command) to eliminate directories that have no applicable files. This hashing can be disabled with the **–c** or **–t**, options, or the **unhash** built-in command.

As a special case, if there is no **/** in the name of the script and there is an alias for the word **shell**, the expansion of the **shell** alias is prepended (without modification) to the command line. The system attempts to execute the first word of this special (late-occurring) alias, which should be a full pathname. Remaining words of the alias's definition, along with the text of the input line, are treated as arguments.

When a pathname is found that has proper execute permissions, the shell forks a new process and passes it, along with its arguments, to the kernel using the **execve**( ) system call (see **exec**(2)). The kernel then attempts to overlay the new process with the desired program. If the file is an executable binary (in **a.out**(4) format) the kernel succeeds and begins executing the new process. If the file is a text file and the first line begins with **#!**, the next word is taken to be the pathname of a shell (or command) to interpret that script. Subsequent words on the first line are taken as options for that shell. The kernel invokes (overlays) the indicated shell, using the name of the script as an argument.

If neither of the above conditions holds, the kernel cannot overlay the file and the **execve**( ) call fails (see **exec**(2)); the C shell then attempts to execute the file by spawning a new shell, as follows:

- If the first character of the file is a **#**, a C shell is invoked.
- Otherwise, a Bourne shell is invoked.

Signal Handling

The shell normally ignores QUIT signals. Background jobs are immune to signals generated from the keyboard, including hangups (HUP). Other signals have the values that the C shell inherited from its environment. The shell's handling of interrupt and terminate signals within scripts can be controlled by the **onintr** built-in command. Login shells catch the TERM signal; otherwise, this signal is passed on to child processes. In no case are interrupts allowed when a login shell is reading the **.logout** file.

Job Control

The shell associates a numbered *job* with each command sequence to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically CTRL-z). When a command or command sequence (semicolon separated list) is started in the background using the **&** metacharacter, the shell displays a line with the job number in brackets and a list of associated process numbers:

```
[1] 1234
```

To see the current list of jobs, use the **jobs** built-in command. The job most recently stopped (or put into the background if none are stopped) is referred to as the *current* job and is indicated with a '**+**'. The previous job is indicated with a '–'; when the current job is terminated or moved to the foreground, this job takes its place (becomes the new current job).

To manipulate jobs, refer to the **bg**, **fg**, **kill**, **stop**, and **%** built-in commands.

A reference to a job begins with a ‘**%**’. By itself, the percent-sign refers to the current job.

% %+ %%	The current job.
%–	The previous job.
%*j*	Refer to job *j* as in: ‘**kill –9 %***j*’. *j* can be a job number, or a string that uniquely specifies the command line by which it was started; ‘**fg %vi**’ might bring a stopped **vi** job to the foreground, for instance.
%?*string*	Specify the job for which the command line uniquely contains *string*.

A job running in the background stops when it attempts to read from the terminal. Background jobs can normally produce output, but this can be suppressed using the ‘**stty tostop**’ command.

Status Reporting

While running interactively, the shell tracks the status of each job and reports whenever the job finishes or becomes blocked. It normally displays a message to this effect as it issues a prompt, in order to avoid disturbing the appearance of your input. When set, the **notify** variable indicates that the shell is to report status changes immediately. By default, the **notify** command marks the current process; after starting a background job, type **notify** to mark it.

Built-In Commands

Built-in commands are executed within the C shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

: Null command. This command is interpreted, but performs no action.

alias [*name* [*def*]]
Assign *def* to the alias *name*. *def* is a list of words that may contain escaped history-substitution metasyntax. *name* is not allowed to be **alias** or **unalias**. If *def* is omitted, the alias *name* is displayed along with its current definition. If both *name* and *def* are omitted, all aliases are displayed.

bg [**%***job* . . .]
Run the current or specified jobs in the background.

break Resume execution after the **end** of the nearest enclosing **foreach** or **while** loop. The remaining commands on the current line are executed. This allows multilevel breaks to be written as a list of **break** commands, all on one line.

breaksw Break from a **switch**, resuming after the **endsw**.

case *label*: A label in a **switch** statement.

cd [*dir*]
chdir [*dir*]
Change the shell’s working directory to directory *dir*. If no argument is given, change to the home directory of the user. If *dir* is a relative pathname not found in the current directory, check for it in those directories listed in the **cdpath** variable. If *dir* is the name of a shell variable whose value starts with a **/**, change to the directory named by that value.

continue Continue execution of the next iteration of the nearest enclosing **while** or **foreach** loop.

default: Labels the default case in a **switch** statement. The default should come after all **case** labels. Any remaining commands on the command line are first executed.

dirs [**–l**]

Print the directory stack, most recent to the left; the first directory shown is the current directory. With the **–l** argument, produce an unabbreviated printout; use of the ˜ notation is suppressed.

echo [**–n**] *list*

The words in *list* are written to the shell's standard output, separated by space characters. The output is terminated with a newline unless the **–n** option is used.

eval *argument* . . .

Reads the arguments as input to the shell and executes the resulting command(s). This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See **tset**(1B) for an example of how to use **eval**.

exec *command*

Execute *command* in place of the current shell, which terminates.

exit [**(***expr***)**]

The shell exits, either with the value of the status variable or with the value specified by the expression *expr* .

fg [**%***job*]

Bring the current or specified *job* into the foreground.

foreach *var* **(***wordlist***)**

. . .

end The variable *var* is successively set to each member of *wordlist*. The sequence of commands between this command and the matching **end** is executed for each new value of *var*. Both **foreach** and **end** must appear alone on separate lines.

The built-in command **continue** may be used to terminate the execution of the current iteration of the loop and the built-in command **break** may be used to terminate execution of the **foreach** command. When this command is read from the terminal, the loop is read once prompting with **?** before any statements in the loop are executed.

glob *wordlist*

Perform filename expansion on *wordlist*. Like **echo**, but no \ escapes are recognized. Words are delimited by **NULL** characters in the output.

goto *label* The specified *label* is a filename and a command expanded to yield a label. The shell rewinds its input as much as possible and searches for a line of the form *label***:** possibly preceded by space or tab characters. Execution continues after the indicated line. It is an error to jump to a label that occurs between a **while** or **for** built-in command and its corresponding **end**.

hashstat Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **exec**s). An **exec** is attempted for each component of the *path* where the hash function indicates a possible hit and in each component that does not begin with a '/'.

history [–**hr**] [*n*]

Display the history list; if *n* is given, display only the *n* most recent events.

–**r** Reverse the order of printout to be most recent first rather than oldest first.

–**h** Display the history list without leading numbers. This is used to produce files suitable for sourcing using the –**h** option to *source*.

if (*expr***)** *command*

If the specified expression evaluates to true, the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Note: I/O redirection occurs even if *expr* is false, when *command* is *not* executed (this is a bug).

if (*expr***) then**
. . .
else if (*expr2***) then**
. . .
else
. . .
endif If *expr* is true, commands up to the first **else** are executed. Otherwise, if *expr2* is true, the commands between the **else if** and the second **else** are executed. Otherwise, commands between the **else** and the **endif** are executed. Any number of **else if** pairs are allowed, but only one **else**. Only one **endif** is needed, but it is required. The words **else** and **endif** must be the first nonwhite characters on a line. The **if** must appear alone on its input line or after an **else**.

jobs[–**l**] List the active jobs under job control.

–**l** List process IDs, in addition to the normal information.

kill [–*sig*] [*pid*] [**%***job*] . . .
kill –l Send the **TERM** (terminate) signal, by default, or the signal specified, to the specified process ID, the *job* indicated, or the current *job*. Signals are either given by number or by name. There is no default. Typing **kill** does not send a signal to the current job. If the signal being sent is **TERM** (terminate) or **HUP** (hangup), then the job or process is sent a **CONT** (continue) signal as well.

–**l** List the signal names that can be sent.

limit [–**h**] [*resource* [*max-use*]]

Limit the consumption by the current process or any process it spawns, each not to exceed *max-use* on the specified *resource*. If *max-use* is omitted, print the current limit; if *resource* is omitted, display all limits.

–**h** Use hard limits instead of the current limits. Hard limits impose a ceiling on the values of the current limits. Only the privileged user may raise the hard limits.

resource is one of:

cputime	Maximum CPU seconds per process.
filesize	Largest single file allowed.
datasize	Maximum data size (including stack) for the process.
stacksize	Maximum stack size for the process.
coredumpsize	Maximum size of a core dump (file).
descriptors	Maximum number of file descriptors.
memorysize	Maximum size of virtual memory.

max-use is a number, with an optional scaling factor, as follows:

*n***h**	Hours (for **cputime**).
*n***k**	*n* kilobytes. This is the default for all but **cputime**.
*n***m**	*n* megabytes or minutes (for **cputime**).
*mm***:***ss*	Minutes and seconds (for **cputime**).

login [*username* | –**p**]

Terminate a login shell and invoke **login**(1). The **.logout** file is not processed. If *username* is omitted, **login** prompts for the name of a user.

–**p** Preserve the current environment (variables).

logout Terminate a login shell.

nice [+*n* | –*n*] [*command*]

Increment the process priority value for the shell or for *command* by *n*. The higher the priority value, the lower the priority of a process, and the slower it runs. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** commands apply. If *command* is omitted, **nice** increments the value for the current shell. If no increment is specified, **nice** sets the process priority value to 4. The range of process priority values is from –20 to 20. Values of *n* outside this range set the value to the lower, or to the higher boundary, respectively.

+*n* Increment the process priority value by *n*.

–*n* Decrement by *n*. This argument can be used only by the privileged user.

nohup [*command*]

Run *command* with HUPs ignored. With no arguments, ignore HUPs throughout the remainder of a script. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** statements apply. All processes detached with **&** are effectively **nohup**’d.

notify [**%***job*] . . .
Notify the user asynchronously when the status of the current job or specified jobs changes.

onintr [– | *label*]
Control the action of the shell on interrupts. With no arguments, **onintr** restores the default action of the shell on interrupts. (The shell terminates shell scripts and returns to the terminal command input level). With the – argument, the shell ignores all interrupts. With a *label* argument, the shell executes a **goto** *label* when an interrupt is received or a child process terminates because it was interrupted.

popd [**+***n*] Pop the directory stack and **cd** to the new top directory. The elements of the directory stack are numbered from 0 starting at the top.

+*n*	Discard the *n*'th entry in the stack.

pushd [**+***n* | *dir*]
Push a directory onto the directory stack. With no arguments, exchange the top two elements.

+*n*	Rotate the *n*'th entry to the top of the stack and **cd** to it.
dir	Push the current working directory onto the stack and change to *dir*.

rehash Recompute the internal hash table of the contents of directories listed in the *path* variable to account for new commands added.

repeat *count command*
Repeat *command count* times. *command* is subject to the same restrictions as with the one-line **if** statement.

set [*var* [= *value*]]
set *var***[***n***]** = *word*
With no arguments, **set** displays the values of all shell variables. Multiword values are displayed as a parenthesized list. With the *var* argument alone, **set** assigns an empty (null) value to the variable *var*. With arguments of the form *var* = *value* **set** assigns *value* to *var*, where *value* is one of:

word	A single word (or quoted string).
(*wordlist***)**	A space-separated list of words enclosed in parentheses.

Values are command and filename expanded before being assigned. The form **set** *var***[***n***]** = *word* replaces the *n*'th word in a multiword value with *word*.

setenv [*VAR* [*word*]]
With no arguments, **setenv** displays all environment variables. With the *VAR* argument, **setenv** sets the environment variable *VAR* to have an empty (null) value. (By convention, environment variables are normally given upper-case names.) With both *VAR* and *word* arguments, **setenv** sets the environment variable **NAME** to the value *word*, which must be either a single word or a quoted string. The most commonly used environment variables, **USER**, **TERM**, and **PATH**, are automatically imported to and exported from the **csh** variables **user**, **term**, and **path**; there is no need to use **setenv** for these. In

addition, the shell sets the **PWD** environment variable from the **csh** variable **cwd** whenever the latter changes.

The environment variables **LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY** take immediate effect when changed within the C shell.

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **csh** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **csh** behaves.

LC_CTYPE

Determines how **csh** handles characters. When **LC_CTYPE** is set to a valid value, **csh** can display and handle text and filenames containing valid characters for that locale. **csh** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **csh** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. / English).

LC_NUMERIC

Determines the value of the radix character (decimal point (".") in the "C" locale) and thousand separator (empty string ("") in the "C" locale).

shift [*variable*]

The components of **argv**, or *variable*, if supplied, are shifted to the left, discarding the first component. It is an error for the variable not to be set or to have a null value.

source [**–h**] *name*

Reads commands from *name*. **source** commands may be nested, but if they are nested too deeply the shell may run out of file descriptors. An error in a sourced file at any level terminates all nested **source** commands.

–h Place commands from the file *name* on the history list without executing them.

stop %*jobid* . . .

Stop the current or specified background job.

stop *pid* . . .

Stop the specified process, *pid*. (see **ps**(1)).

suspend Stop the shell in its tracks, much as if it had been sent a stop signal with **ˆZ**. This is most often used to stop shells started by **su**.

switch (*string***)**
case *label***:**
. . .
breaksw
. . .
default:
. . .
breaksw
endsw

Each *label* is successively matched, against the specified *string*, which is first command and filename expanded. The file metacharacters ∗, **?** and **[**. . .**]** may be used in the case labels, which are variable expanded. If none of the labels match before a "default" label is found, execution begins after the default label. Each **case** statement and the **default** statement must appear at the beginning of a line. The command **breaksw** continues execution after the **endsw**. Otherwise control falls through subsequent **case** and **default** statements as with C. If no label matches and there is no default, execution continues after the **endsw**.

time [*command*]

With no argument, print a summary of time used by this C shell and its children. With an optional *command*, execute *command* and print a summary of the time it uses.

umask [*value*]

Display the file creation mask. With *value*, set the file creation mask. With *value* given in octal, the user can turn-off any bits, but cannot turn-on bits to allow new permissions. Common values include 077, restricting all permissions from everyone else; 002, giving complete access to the group, and read (and directory search) access to others; or 022, giving read (and directory search) but not write permission to the group and others.

unalias *pattern*

Discard aliases that match (filename substitution) *pattern*. All aliases are removed by '**unalias** ∗'.

unhash Disable the internal hash table.

unlimit [–**h**] [*resource*]

Remove a limitation on *resource*. If no *resource* is specified, then all resource limitations are removed. See the description of the **limit** command for the list of resource names.

–**h** Remove corresponding hard limits. Only the privileged user may do this.

unset *pattern*

Remove variables whose names match (filename substitution) *pattern*. All variables are removed by '**unset** ∗'; this has noticeably distasteful side effects.

unsetenv *variable*

Remove *variable* from the environment. As with **unset**, pattern matching is not performed.

wait Wait for background jobs to finish (or for an interrupt) before prompting.

while (*expr***)**
. . .
end While *expr* is true (evaluates to nonzero), repeat commands between the **while** and the matching **end** statement. **break** and **continue** may be used to terminate or continue the loop prematurely. The **while** and **end** must appear alone on their input lines. If the shell's input is a terminal, it prompts for commands with a question-mark until the **end** command is entered and then performs the commands in the loop.

%[*job*] [**&**]

Bring the current or indicated *job* to the foreground. With the ampersand, continue running *job* in the background.

@ [*var* **=***expr*]
@ [*var***[***n***]** **=***expr*]

With no arguments, display the values for all shell variables. With arguments, set the variable *var*, or the *n*'th word in the value of *var*, to the value that *expr* evaluates to. (If **[***n***]** is supplied, both *var* and its *n*'th component must already exist.)

If the expression contains the characters **>**, **<**, **&**, or **|**, then at least this part of *expr* must be placed within parentheses.

The operators ∗**=**, **+=**, and so forth, are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* that would otherwise be single words.

Special postfix operators, **+ +** and – –, increment or decrement *name*, respectively.

Environment Variables and Predefined Shell Variables

Unlike the Bourne shell, the C shell maintains a distinction between environment variables, which are automatically exported to processes it invokes, and shell variables, which are not. Both types of variables are treated similarly under variable substitution. The shell sets the variables **argv**, **cwd**, **home**, **path**, **prompt**, **shell**, and **status** upon initialization. The shell copies the environment variable **USER** into the shell variable **user**, **TERM** into **term**, and **HOME** into **home**, and copies each back into the respective environment variable whenever the shell variables are reset. **PATH** and **path** are similarly handled. You need only set **path** once in the **.cshrc** or **.login** file. The environment variable **PWD** is set from **cwd** whenever the latter changes. The following shell variables have predefined meanings:

argv	Argument list. Contains the list of command line arguments supplied to the current invocation of the shell. This variable determines the value of the positional parameters **$1**, **$2**, and so on.
cdpath	Contains a list of directories to be searched by the **cd**, **chdir**, and **popd** commands, if the directory argument each accepts is not a subdirectory of the current directory.
cwd	The full pathname of the current directory.
echo	Echo commands (after substitutions) just before execution.
fignore	A list of filename suffixes to ignore when attempting filename completion. Typically the single word '**.o**'.
filec	Enable filename completion, in which case the CTRL-d character EOT and the ESC character have special significance when typed in at the end of a terminal input line: EOT Print a list of all filenames that start with the preceding string. ESC Replace the preceding string with the longest unambiguous extension.
hardpaths	If set, pathnames in the directory stack are resolved to contain no symbolic-link components.
histchars	A two-character string. The first character replaces **!** as the history-substitution character. The second replaces the carat (ˆ) for quick substitutions.
history	The number of lines saved in the history list. A very large number may use up all of the C shell's memory. If not set, the C shell saves only the most recent command.
home	The user's home directory. The filename expansion of **~** refers to the value of this variable.
ignoreeof	If set, the shell ignores EOF from terminals. This protects against accidentally killing a C shell by typing a CTRL-d.

mail	A list of files where the C shell checks for mail. If the first word of the value is a number, it specifies a mail checking interval in seconds (default 5 minutes).
nobeep	Suppress the bell during command completion when asking the C shell to extend an ambiguous filename.
noclobber	Restrict output redirection so that existing files are not destroyed by accident. **>** redirections can only be made to new files. **>>** redirections can only be made to existing files.
noglob	Inhibit filename substitution. This is most useful in shell scripts once filenames (if any) are obtained and no further expansion is desired.
nonomatch	Returns the filename substitution pattern, rather than an error, if the pattern is not matched. Malformed patterns still result in errors.
notify	If set, the shell notifies you immediately as jobs are completed, rather than waiting until just before issuing a prompt.
path	The list of directories in which to search for commands. **path** is initialized from the environment variable **PATH**, which the C shell updates whenever **path** changes. A null word specifies the current directory. The default is typically: **(/bin /usr/bin /usr/ucb /etc .)**. If **path** becomes unset only full pathnames will execute. An interactive C shell will normally hash the contents of the directories listed after reading **.cshrc**, and whenever **path** is reset. If new commands are added, use the **rehash** command to update the table.
prompt	The string an interactive C shell prompts with. Noninteractive shells leave the **prompt** variable unset. Aliases and other commands in the **.cshrc** file that are only useful interactively, can be placed after the following test: '**if ($?prompt == 0) exit**', to reduce startup time for noninteractive shells. A **!** in the **prompt** string is replaced by the current event number. The default prompt is *hostname***%** for mere mortals, or *hostname***#** for the privileged user.
savehist	The number of lines from the history list that are saved in **~/.history** when the user logs out. Large values for **savehist** slow down the C shell during startup.
shell	The file in which the C shell resides. This is used in forking shells to interpret files that have execute bits set, but that are not executable by the system.
status	The status returned by the most recent command. If that command terminated abnormally, 0200 is added to the status. Built-in commands that fail return exit status 1; all other built-in commands set status to 0.
time	Control automatic timing of commands. Can be supplied with one or two values. The first is the reporting threshold in CPU seconds. The second is a string of tags and text indicating which resources to

report on. A tag is a percent sign (**%**) followed by a single uppercase letter (unrecognized tags print as text):

%D	Average amount of unshared data space used in Kilobytes.
%E	Elapsed (wallclock) time for the command.
%F	Page faults.
%I	Number of block input operations.
%K	Average amount of unshared stack space used in Kilobytes.
%M	Maximum real memory used during execution of the process.
%O	Number of block output operations.
%P	Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time.
%S	Number of seconds of CPU time consumed by the kernel on behalf of the user’s process.
%U	Number of seconds of CPU time devoted to the user’s process.
%W	Number of swaps.
%X	Average amount of shared memory used in Kilobytes.

The default summary display outputs from the **%U**, **%S**, **%E**, **%P**, **%X**, **%D**, **%I**, **%O**, **%F**, and **%W** tags, in that order.

verbose Display each command after history substitution takes place.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shells after **.cshrc** at login.
~/.logout	Read by login shells at logout.
~/.history	Saved history for use at next login.
/usr/bin/sh	The Bourne shell, for shell scripts not starting with a ‘**#**’.
/tmp/sh∗	Temporary file for ‘<<’.
/etc/passwd	Source of home directories for ‘~*name*’.

SEE ALSO

login(1), **ps**(1), **sh**(1), **shell_builtins**(1), **tset**(1B), **access**(2), **exec**(2), **fork**(2), **pipe**(2), **a.out**(4), **environ**(4), **environ**(5), **ascii**(5), **termio**(7)

DIAGNOSTICS

You have stopped jobs.

You attempted to exit the C shell with stopped jobs under job control. An immediate second attempt to exit will succeed, terminating the stopped jobs.

NOTES

Words can be no longer than 1024 characters. The system limits argument lists to 1,048,576 characters. However, the maximum number of arguments to a command for which filename expansion applies is 1706. Command substitutions may expand to no more characters than are allowed in the argument list. To detect looping, the shell restricts the number of **alias** substitutions on a single line to 20.

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (that is, wrong) as the job may have changed directories internally.

Shell built-in functions are not stoppable / restartable. Command sequences of the form *a* **;** *b* **;** *c* are also not handled gracefully when stopping is attempted. If you suspend *b*, the shell never executes *c*. This is especially noticeable if the expansion results from an alias. It can be avoided by placing the sequence in parentheses to force it into a subshell.

Control over terminal output after processes are started is primitive; use the Sun Window system if you need better output control.

Multiline shell procedures should be provided, as they are with the Bourne shell.

Commands within loops, prompted for by **?**, are not placed in the *history* list.

Control structures should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with **|**, and to be used with **&** and **;** metasyntax.

It should be possible to use the **:** modifiers on the output of command substitutions. There are two problems with **:** modifier usage on variable substitutions: not all of the modifiers are available, and only one modifier per substitution is allowed.

The **g** (global) flag in history substitutions applies only to the first match in each word, rather than all matches in all words. The common text editors consistently do the latter when given the **g** flag in a substitution command.

Quoting conventions are confusing. Overriding the escape character to force variable substitutions within double quotes is counterintuitive and inconsistent with the Bourne shell.

Symbolic links can fool the shell. Setting the **hardpaths** variable alleviates this.

‘**set path**’ should remove duplicate pathnames from the pathname list. These often occur because a shell script or a **.cshrc** file does something like ‘**set path=(/usr/local /usr/hosts $path)**’ to ensure that the named directories are in the pathname list.

The only way to direct the standard output and standard error separately is by invoking a subshell, as follows:

```
example% (command > outfile) >& errorfile
```

Although robust enough for general use, adventures into the esoteric periphery of the C shell may reveal unexpected quirks.

If you start **csh** as a login shell and you do not have a **.login** in your home directory, then the **csh** reads in the **/etc/.login**.

NAME

csplit – split a file with respect to a given context

SYNOPSIS

csplit [**–s**] [**–k**] [**–f** *prefix*] *filename argument1* [... *argumentn*]

AVAILABILITY

SUNWesu

DESCRIPTION

csplit reads *filename* and separates it into *n*+1 sections, defined by the arguments *argument1* ... *argumentn*. By default the sections are placed in **xx00** ... **xx***n* (*n* may not be greater than 99). These sections get the following pieces of *filename*:

00:	From the start of *filename* up to (but not including) the line referenced by *argument1*.
01:	From the line referenced by *argument1* up to the line referenced by *argument2*.
	•
	•
	•
n:	From the line referenced by *argumentn* to the end of *filename*.

If the *filename* argument is a –, then standard input is used.

The arguments (*argument1* ... *argumentn*) to **csplit** can be a combination of the following:

/*rexp*/	A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional **+** or – some number of lines (for example, **/Page/–5**). See **ed**(1) for a description of how to specify a regular expression.
%*rexp*%	This argument is the same as /*rexp*/, except that no file is created for the section.
lnno	A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.
{*num*}	Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded newlines. **csplit** does not affect the original file; it is the user's responsibility to remove it if it is no longer wanted.

OPTIONS

–s	**csplit** normally prints the character counts for each file created. If the **–s** option is present, **csplit** suppresses the printing of all character counts.
–k	**csplit** normally removes created files if an error occurs. If the **–k** option is present, **csplit** leaves previously created files intact.
–f *prefix*	If the **–f** option is used, the created files are named *prefix***00...***prefixn*. The default is **xx00** ... **xx***n*.

EXAMPLES

This example creates four files, **cobol00** . . . **cobol03**.

```
example% csplit –f cobol filename ’/procedure division/’ /par5./ /par16./
```

After editing the ‘‘split’’ files, they can be recombined as follows:

```
example% cat cobol0[0–3] > filename
```

Note: This example overwrites the original file.

This example splits the file at every 100 lines, up to 10,000 lines. The **–k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
example% csplit –k filename 100 {99}
```

If **prog.c** follows the normal C coding convention (the last line of a routine consists only of a **}** in the first character position), this example creates a file for each separate C routine (up to 21) in **prog.c**.

```
example% csplit –k prog.c ’%main(%´ ’/^}/+1’ {20}
```

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **csplit** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **csplit** behaves.

LC_CTYPE

Determines how **csplit** handles characters. When **LC_CTYPE** is set to a valid value, **csplit** can display and handle text and filenames containing valid characters for that locale. **csplit** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **csplit** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

ed(1), **sh**(1), **environ**(5)

DIAGNOSTICS

Self-explanatory except for:

arg – **out of range**

which means that the given argument did not reference a line between the current position and the end of the file.

NAME

ct – spawn login to a remote terminal

SYNOPSIS

ct [*options*] *telno* . . .

AVAILABILITY

SUNWbnuu

DESCRIPTION

ct dials the telephone number of a modem that is attached to a terminal and spawns a **login** process to that terminal. The *telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. (The set of legal characters for *telno* is 0 through 9, **-**, =, ∗, and #. The maximum length *telno* is 31 characters). If more than one telephone number is specified, **ct** will try each in succession until one answers; this is useful for specifying alternate dialing paths.

ct will try each line listed in the file **/etc/uucp/Devices** until it finds an available line with appropriate attributes, or runs out of entries.

After the user on the destination terminal logs out, there are two things that could occur depending on what type of port monitor is monitoring the port. In the case of no port monitor, **ct** prompts: **Reconnect?** If the response begins with the letter **n**, the line will be dropped; otherwise, *ttymon* will be started again and the **login:** prompt will be printed. In the second case, where a port monitor is monitoring the port, the port monitor reissues the **login:** prompt.

The user should log out properly before disconnecting.

OPTIONS

–**h** Normally, **ct** will hang up the current line so that it can be used to answer the incoming call. The –**h** option will prevent this action. The –**h** option will also wait for the termination of the specified **ct** process before returning control to the user's terminal.

–**s***speed* The data rate may be set with the –**s** option. *speed* is expressed in baud rates. The default baud rate is 1200.

–**v** If the –**v** (verbose) option is used, **ct** will send a running narrative to the standard error output stream.

–**w***n* If there are no free lines **ct will ask if it should** if so, for how many minutes it should wait before it gives up. **ct** will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. This dialogue may be overridden by specifying the –**w***n* option. *n* is the maximum number of minutes that **ct** is to wait for a line.

–**x***n* This option is used for debugging; it produces a detailed output of the program execution on stderr. *n* is a single number between 0 and 9. As *n* increases to 9, more detailed debugging information is given.

FILES

/etc/uucp/Devices
/var/adm/ctlog

SEE ALSO

cu(1C), **login**(1), **uucp**(1C), **ttymon**(1M)

NOTES

The **ct** program will not work with a DATAKIT Multiplex interface.

For a shared port, one used for both dial-in and dial-out, the *ttymon* program running on the line must have the –**r** and –**b** options specified (see **ttymon**(1M)).

NAME

ctags – create a tags file for use with ex and vi

SYNOPSIS

ctags [**–aBFtuvwx**] [**–f** *tagsfile*] *filename*. . .

DESCRIPTION

ctags makes a tags file for **ex**(1) from the specified C, C++, Pascal, FORTRAN, YACC, and LEX sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by SPACE or TAB characters. Using the tags file, **ex** can quickly find these objects definitions.

Normally **ctags** places the tag descriptions in a file called **tags**; this may be overridden with the **–f** option.

Files with names ending in **.c** or **.h** are assumed to be either C or C++ source files and are searched for C / C++ routine and macro definitions. Files with names ending in **.cc**, **.C**, or **.cxx**, are assumed to be C++ source files. Files with names ending in **.y** are assumed to be YACC source files. Files with names ending in **.l** are assumed to be LEX files. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C definitions.

The tag **main** is treated specially in C or C++ programs. The tag formed is created by prepending **M** to *filename*, with a trailing **.c** , **.cc** **.C**, or **.cxx** removed, if any, and leading pathname components also removed. This makes use of **ctags** practical in directories with more than one program.

OPTIONS

–a	Append output to an existing **tags** file.
–B	Use backward searching patterns (**?**. . . **?**).
–F	Use forward searching patterns (**/**. . . **/**) (default).
–t	Create tags for typedefs.
–u	Update the specified files in tags, that is, all references to them are deleted, and the new values are appended to the file. Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the **tags** file.
–v	Produce on the standard output an index listing the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through **sort –f**.
–w	Suppress warning diagnostics.
–x	Produce a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.
–f *tagsfile*	Places the tag descriptions in a file called *tagsfile* instead of **tags**.

USAGE

The **–v** option is mainly used with **vgrind** which will be part of the optional BSD Compatibility Package.

EXAMPLES

Using **ctags** with the **–v** option produces entries in an order which may not always be appropriate for **vgrind**. To produce results in alphabetical order, you may want to run the output through '**sort –f**'.

```
example% ctags –v filename.c filename.h | sort –f > index
example% vgrind –x index
```

To build a tags file for C sources in a directory hierarchy rooted at *sourcedir*, first create an empty tags file, and then run **find**(1):

```
example% cd sourcedir ; rm –f tags ; touch tags
example% find . \( –name SCCS –prune –name \
        '*.c' –o –name '*.h' \) –exec ctags –u {} \;
```

Note that spaces must be entered exactly as shown.

FILES

tags	output tags file

SEE ALSO

ex(1), **vgrind**(1), **vi**(1)

NOTES

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

ctags does not know about **#ifdefs**.

ctags should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **–tx** shows only the last line of typedefs.

NAME

cu – call another UNIX system

SYNOPSIS

cu [**–c** *device* | **–l** *line*] [**–s** *speed*] [**–b** *bits*] [**–h**] [**–n**] [**–t**] [**–d**] [**–o** | **–e**]
[**–L**] [**–C**] [**–H**] *telno* | *systemname* [*local-cmd*]

AVAILABILITY

SUNWbnuu

DESCRIPTION

cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of files. It is convenient to think of **cu** as operating in two phases. The first phase is the connection phase in which the connection is established. **cu** then enters the conversation phase. The **–d** option is the only one that applies to both phases.

OPTIONS

cu accepts many options. The **–c**, **–l**, and **–s** options play a part in selecting the medium; the remaining options are used in configuring the line.

–c *device*	The first field in the **/etc/uucp/Devices** file is the "Type" field. The **–c** option forces **cu** to only use entries in the "Type" field that match the user specified *device*. The specified *device* is usually the name of a local area network.
–s *speed*	Specifies the transmission speed (**300**, **1200**, **2400 9600**, **19200**, **38400**). The default value is "Any" speed which will depend on the order of the lines in the **/etc/uucp/Devices** file.
–l *line*	Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the **–l** option is used without the **–s** option, the speed of a line is taken from the **/etc/uucp/Devices** file record in which **line** matches the second field (the Line field). When the **–l** and **–s** options are both used together, **cu** will search the **/etc/uucp/Devices** file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed, otherwise, an error message will be printed and the call will not be made. In the general case where a specified device is a directly connected asynchronous line (for instance, **/dev/term/a**), a telephone number (*telno*) is not required. The specified device need not be in the **/dev** directory. If the specified device is associated with an auto dialer, a telephone number must be provided.
–b *bits*	Forces *bits* to be the number of bits processed on the line. *bits* is either **7** or **8**. This allows connection between systems with different character sizes. By default, the character size of the line is set to the same as the current local terminal.
–h	Set communication mode to half-duplex. This option emulates local echo in order to support calls to other computer systems that expect terminals to be set to half-duplex mode.

–n	Request user prompt for telephone number. For added security, this option will prompt the user to provide the telephone number to be dialed, rather than taking it from the command line.
–t	Used to dial a terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.
–d	Diagnostic traces are printed.
–o	Set an ODD data parity. This option designates that ODD parity is to be generated for data sent to the remote system.
–e	Set an EVEN data parity. This option designates that EVEN parity is to be generated for data sent to the remote system.
–L	will cause **cu** to go through the login chat sequence specified in the **/etc/uucp/Systems** file. For more information about the chat sequence, see *TCP/IP Network Administration Guide*
–C	will cause **cu** to run the local command that is specified at the end of the command line, instead of entering interactive mode. The **stdin** and **stdout** of the command that is run refer to the remote connection.
–H	ignore one hangup. This allows the user to remain in **cu** while the remote machine disconnects and places a call back to the local machine. This option should be used when connecting to systems with callback or dialback modems. Once the callback occurs subsequent hangups will cause **cu** to terminate. This option can be specified more than once. For more information about dialback configuration, see **remote**(4) and *TCP/IP Network Administration Guide*

USAGE

Connection Phase

cu uses the same mechanism that **uucp**(1C) does to establish a connection. This means that it will use the **uucp** control files **/etc/uucp/Devices** and **/etc/uucp/Systems**. This gives **cu** the ability to choose from several different media to establish the connection. The possible media include telephone lines, direct connections, and local area networks (LAN). The **/etc/uucp/Devices** file contains a list of media that are available on your system. The **/etc/uucp/Systems** file contains information for connecting to remote systems, but it is not generally readable.

Note: **cu** determines which **/etc/uucp/Systems** and **/etc/uucp/Devices** files to use based upon the name used to invoke **cu**. In the simple case, this name will be "cu", but you could also have created a link to **cu** with another name, e.g.,"pppcu", in which case **cu** would then look for a "service=pppcu" entry in the **/etc/uucp/Sysfiles** file, which will tell it which **/etc/uucp/Systems** file to use.

The *telno* or *systemname* parameter from the command line is used to tell **cu** what system you wish to connect to. This parameter can be blank, a telephone number, a system name, or a LAN specific address.

telephone number	A telephone number is a string consisting of the tone dial characters (the digits **0** through **9**, ∗, and **#**) plus the special characters **=** and –. The equal sign designates a secondary dial tone and the

	minus sign creates a **4** second delay.
system name	A system name is the name of any computer that **uucp** can call; the **uuname**(1C) command prints a list of these names.
LAN address	The documentation for your LAN will show the form of the LAN specific address.

If **cu**'s default behavior is invoked (not using the –**c** or –**l** options), **cu** will use the *telno* or *systemname* parameter to determine which medium to use. If a telephone number is specified, **cu** will assume that you wish to use a telephone line and it will select an automatic call unit (ACU). Otherwise, **cu** will assume that it is a system name. **cu** will follow the **uucp** calling mechanism and use the **/etc/uucp/Systems** and **/etc/uucp/Devices** files to obtain the best available connection. Since **cu** will choose a speed that is appropriate for the medium that it selects, you may not use the –**s** option when this parameter is a system name.

The –**c** and –**l** options modify this default behavior. –**c** is most often used to select a LAN by specifying a Type field from the **/etc/uucp/Devices** file. You must include either a *telno* or *systemname* value when using the –**c** option. If the connection to *systemname* fails, a connection will be attempted using *systemname* as a LAN specific address. The –**l** option is used to specify a device associated with a direct connection. If the connection is truly a direct connection to the remote machine, then there is no need to specify a *systemname*. This is the only case where a *telno* or *systemname* parameter is unnecessary. On the other hand, there may be cases in which the specified device connects to a dialer, so it is valid to specify a telephone number. The –**c** and –**l** options should not be specified on the same command line.

Conversation Phase

After making the connection, **cu** runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings.

Commands

The *transmit* process interprets the following user initiated commands:

~.	Terminate the conversation.
~!	Escape to an interactive shell on the local system.
~!*cmd*. . .	Run *cmd* on the local system (via **sh** –**c**).
~$*cmd*. . .	Run *cmd* locally and send its output to the remote system.
~%cd	Change the directory on the local system. Note: **~!cd** will cause the command to be run by a sub-shell, probably not what was intended.
~%take *from* [*to*]	Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.

~%put *from* [*to*]	Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.
~~ *line*	Send the line **~ line** to the remote system.
~%break	Transmit a **BREAK** to the remote system (which can also be specified as **~%b**).
~%debug	Toggles the **–d** debugging option on or off (which can also be specified as **~%d**).
~t	Prints the values of the termio structure variables for the user's terminal (useful for debugging).
~l	Prints the values of the termio structure variables for the remote communication line (useful for debugging).
~%ifc	Toggles between DC3 / DC1 input control protocol and no input control. This is useful when the remote system does not respond properly to the DC3 and DC1 characters (can also be specified as ~**%nostop**).
~%ofc	Toggles the output flow control setting. When enabled, outgoing data may be flow controlled by the remote host (can also be specified as ~**%noostop**).
~%divert	Allow / disallow unsolicited diversions. That is, diversions not specified by **~%take**.
~%old	Allow / disallow old style syntax for received diversions.

The *receive* process normally copies data from the remote system to the standard output of the local system. It may also direct the output to local files.

The use of **~%put** requires **stty**(1) and **cat**(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of **echo**(1) and **cat**(1) on the remote system, and that the remote system must be using the Bourne shell, **sh**. Also, **tabs** mode (see **stty**(1)) should be set on the remote system if tabs are to be copied without expansion to spaces.

When **cu** is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. Executing a tilde command reminds the user of the local system **uname**. For example, **uname** can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!uname
Y
```

In general, ~ causes the command to be executed on the original machine. ~~ causes the command to be executed on the next machine in the chain.

EXAMPLES

To dial a system whose telephone number is **9 1 201 555 1234** using **1200** baud (where dialtone is expected after the **9**):

```
example% cu –s 1200 9=12015551234
```

If the speed is not specified, "Any" is the default value.

To login to a system connected by a direct line:

```
example% cu –l /dev/term/b
```

or

```
example% cu –l term/b
```

To dial a system with a specific line and speed:

```
example% cu –s 1200 –l term/b
```

To use a system name:

```
example% cu systemname
```

RETURN VALUES

If successful, **cu** returns **0**; otherwise, –**1** is returned.

FILES

/etc/uucp/Devices
/etc/uucp/Sysfiles
/etc/uucp/Systems
/var/spool/locks/∗

SEE ALSO

cat(1), **stty**(1), **uname**(1), **ct**(1C), **uuname**(1C), **uucp**(1C)

TCP/IP Network Administration Guide

NOTES

The **cu** command does not do any integrity checking on data it transfers. Data fields with special **cu** characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a **~**. to terminate the conversion, even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands. **~%put** and **~%take** cannot be used over multiple links. Files must be moved one link at a time.

There is an artificial slowing of transmission by **cu** during the **~%put** operation so that loss of data is unlikely. Files transferred using **~%take** or **~%put** must contain a trailing newline, otherwise, the operation will hang. Entering a CTRL-D command usually clears the hang condition.

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut –**b** *list* [–**n**] [*filename* ...]
cut –**c** *list* [*filename* ...]
cut –**f** *list* [–**d** *delim*] [–**s**] [*filename*]

AVAILABILITY

SUNWcsu

DESCRIPTION

Use **cut** to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, that is, character positions as on a punched card (–**c** option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (–**f** option). **cut** can be used as a filter; if no files are given, the standard input is used. In addition, a file name of **"–"** explicitly refers to standard input.

Either the –**b**, –**c**, or –**f** option must be specified.

Use **grep**(1) to make horizontal ''cuts'' (by context) through a file, or **paste**(1) to put files together column-wise (that is, horizontally). To reorder columns in a table, use **cut** and **paste**.

OPTIONS

list	A comma-separated list of integer field numbers (in increasing order), with optional – to indicate ranges (for instance, **1,4,7**; **1**–**3,8**; –**5,10** (short for **1**–**5,10**); or **3**– (short for third through last field)).
–**b** *list*	The *list* following –**b** specifies byte positions (for instance, –**b1**–**72** would pass the first 72 bytes of each line). When –**b** and –**n** are used together, *list* is adjusted so that no multi-byte character is split. If –**b** is used, the input line should contain 1023 bytes or less.
–**c** *list*	The *list* following –**c** specifies character positions (for instance, –**c1**–**72** would pass the first 72 characters of each line).
–**d** *delim*	The character following –**d** is the field delimiter (–**f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted. *delim* can be a multi-byte character.
–**f** *list*	The *list* following –**f** is a list of fields assumed to be separated in the file by a delimiter character (see –**d**); for instance, –**f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless –**s** is specified. If –**f** is used, the input line should contain 1023 characters or less.
–**n**	Do not split characters. When –**b** *list* and –**n** are used together, *list* is adjusted so that no multi-byte character is split.
–**s**	Suppresses lines with no delimiter characters in case of –**f** option. Unless specified, lines with no delimiters will be passed through untouched.

EXAMPLES

A mapping of user IDs to names follows:

```
example% cut –d: –f1,5 /etc/passwd
```

To set **name** to current login name follows:

```
example% name=`who am i | cut –f1 –d `
```

ENVIRONMENT

If any of the **LC_∗** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **cut** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_∗** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **cut** behaves.

LC_CTYPE

Determines how **cut** handles characters. When **LC_CTYPE** is set to a valid value, **cut** can display and handle text and filenames containing valid characters for that locale. **cut** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **cut** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

grep(1), **paste**(1), **environ**(5)

DIAGNOSTICS

ERROR: line too long

A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c / f option

Missing –**c** or –**f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

ERROR: no fields

The *list* is empty.

ERROR: no delimiter

Missing *char* on –**d** option.

ERROR: no newline in the last
The input file ends without a newline

ERROR: invalid multibyte character
An error in the input data.

WARNING: cannot open <filename>
Either *filename* cannot be read or does not exist. If multiple filenames are present, processing continues.

NAME

date – print and set the date

SYNOPSIS

date [–**u**] [**+** *format*]
date [–**a** [–]*sss***.***fff*]
date [–**u**] [[*mmdd*] *HHMM* | *mmddHHMM* [*cc*] *yy*]

AVAILABILITY

SUNWcsu

DESCRIPTION

If no argument is given, or if the argument begins with **+**, the current date and time are printed. Otherwise, the current date is set (only by super-user).

Specifications of native language translations of month and weekday names are supported. The month and weekday names used for a language are based on the locale specified by the environment variables **LC_TIME** and **LANG** (see **environ**(5)).

The month and weekday names used for a language are taken from a file whose format is specified in **strftime**(4). This file also defines country-specific date and time formats such as **%C**, which specifies the default date format. The following form is the default for **%C**:

%a %b %e %T %Z %Y

for example,

Fri Dec 23 10:10:42 EST 1988

Field Descriptors (must be preceded by a **%**):

a	abbreviated weekday name
A	full weekday name
b	abbreviated month name
B	full month name
c	locale’s appropriate date and time representation
C	default date and time format
d	day of month – 01 to 31
D	date as **%m/%d/%y**
e	day of month – 1 to 31 (single digits are preceded by a blank)
h	abbreviated month name (alias for **%b**)
H	hour – 00 to 23
I	hour – 01 to 12
j	day of year – 001 to 366
m	month of year – 01 to 12
M	minute – 00 to 59
n	insert a new-line character
p	string containing ante-meridiem or post-meridiem indicator (by default, AM or PM)
r	time as **%I:%M:%S %p**
R	time as **%H:%M**
S	second – 00 to 61, allows for leap seconds
t	insert a tab character
T	time as **%H:%M:%S**

U	week number of year (Sunday as the first day of the week) – 00 to 53
w	day of week – Sunday = 0
W	week number of year (Monday as the first day of the week) – 00 to 53
x	Country-specific date format
X	Country-specific time format
y	year within century – 00 to 99
Y	year as *ccyy* (4 digits)
Z	timezone name

OPTIONS

–a [–]*sss***.***fff* Slowly adjust the time by *sss*.*fff* seconds (*fff* represents fractions of a second). This adjustment can be positive or negative. The system's clock will be sped up or slowed down until it has drifted by the number of seconds specified.

–u Display (or set) the date in Greenwich Mean Time (GMT—universal time), bypassing the normal conversion to (or from) local time.

mm	is the month number
dd	is the day number in the month
HH	is the hour number (24 hour system)
MM	is the minute number
cc	is the century minus one
yy	is the last 2 digits of the year number

The month, day, year, and century may be omitted; the current values are applied as defaults. For example:

date 10080045

sets the date to Oct 8, 12:45 AM. The current year is the default because no year is supplied. The system operates in GMT. **date** takes care of the conversion to and from local standard and daylight time. Only the super-user may change the date. After successfully setting the date and time, **date** displays the new date according to the default format. The **date** command uses **TZ** to determine the correct time zone information (see **environ**(5)).

+ *format* If the argument begins with **+**, the output of **date** is under the control of the user. Each Field Descriptor, described above, is preceded by **%** and is replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output without change. The string is always terminated with a new-line character. If the argument contains embedded blanks it must be quoted (see the **EXAMPLES** section).

EXAMPLES

The command

```
example% date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

generates as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE**, **LC_MESSAGES**, **LC_TIME**, **LC_COLLATE**, **LC_NUMERIC**, and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **date** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **date** behaves.

LC_CTYPE
Determines how **date** handles characters. When **LC_CTYPE** is set to a valid value, **date** can display and handle text and filenames containing valid characters for that locale. **date** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **date** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME
Determines how **date** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

SEE ALSO

strftime(4), **environ**(5)

DIAGNOSTICS

No permission	You are not the super-user and you try to change the date.
bad conversion	The date set is syntactically incorrect.

NOTES

If you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

NAME

dc – desk calculator

SYNOPSIS

dc [*filename*]

AVAILABILITY

SUNWesu

DESCRIPTION

dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. **bc** is a preprocessor for **dc** that provides infix notation and a C-like syntax that implements functions. **bc** also provides reasonable control structures for programs. See **bc**(1). The overall structure of **dc** is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input.

USAGE

The following constructions are recognized:

number
The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9. It may be preceded by an underscore (**_**) to input a negative number. Numbers may contain decimal points.

+ – / * % ^
The top two values on the stack are added (**+**), subtracted (**–**), multiplied (*****), divided (**/**), remaindered (**%**), or exponentiated (**^**). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x* The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x* The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged.

P Interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack are printed.

q Exits the program. If executing a string, the recursion level is popped by two.

Q Exits the program. The top value on the stack is popped and the string execution level is popped by that value.

x Treats the top element of the stack as a character string and executes it as a string of **dc** commands.

X Replaces the number on the top of the stack with its scale factor.

[...] Puts the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x*	The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.
v	Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.
!	Interprets the rest of the line as a shell command.
c	All values on the stack are popped.
i	The top value on the stack is popped and used as the number radix for further input. **I** Pushes the input base on the top of the stack.
o	The top value on the stack is popped and used as the number radix for further output.
O	Pushes the output base on the top of the stack.
k	The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
z	The stack level is pushed onto the stack.
Z	Replaces the number on the top of the stack with its length.
?	A line of input is taken from the input source (usually the terminal) and executed.
; :	are used by **bc**(1) for array operations.

EXAMPLES

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1)

DIAGNOSTICS

x **is unimplemented**	*x* is an octal number.
stack empty	Not enough elements on the stack to do what was asked.
Out of space	The free list is exhausted (too many digits).
Out of headers	Too many numbers being kept around.
Out of pushdown	Too many items on the stack.
Nesting Depth	Too many levels of nested execution.

NAME

deroff – remove nroff / troff, tbl, and eqn constructs

SYNOPSIS

deroff [–**m** [**m** | **s** | **l**] [–**w**] [–**i**] [*filename* . . .]

AVAILABILITY

SUNWdoc

DESCRIPTION

deroff reads each of the *filenames* in sequence and removes all **troff**(1) requests, macro calls, backslash constructs, **eqn**(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and **tbl**(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. **deroff** follows chains of included files (**.so** and **.nx troff** commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, **deroff** reads the standard input.

OPTIONS

–**m** The –**m** option may be followed by an **m**, **s**, or **l**. The –**mm** option causes the macros to be interpreted so that only running text is output (that is, no text from macro lines.) The –**ml** option forces the –**mm** option and also causes deletion of lists associated with the **mm** macros.

–**w** If the –**w** option is given, the output is a word list, one ‘‘word’’ per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a ‘‘word’’ is any string that *contains* at least two letters and is composed of letters, digits, ampersands (**&**), and apostrophes (**′**); in a macro call, however, a ‘‘word’’ is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from ‘‘words.’’

–**i** The –**i** option causes **deroff** to ignore **.so** and **.nx** commands.

SEE ALSO

eqn(1), **nroff**(1), **tbl**(1), **troff**(1)

NOTES

deroff is not a complete **troff** interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The –**ml** option does not handle nested lists correctly.

NAME

df – display status of disk space on file systems

SYNOPSIS

/usr/ucb/df [**–a**] [**–i**] [**–t** *type*] [*filesystem*...] [*filename*...]

AVAILABILITY

SUNWscpu

DESCRIPTION

df displays the amount of disk space occupied by currently mounted file systems, the amount of used and available space, and how much of the file system's total capacity has been used.

If arguments to **df** are path names, **df** produces a report on the file system containing the named file. Thus '**df .**' shows the amount of space on the file system containing the current directory.

OPTIONS

–a Report on all filesystems including the uninteresting ones which have zero total blocks. (that is, auto- mounter)

–i Report the number of used and free inodes. Print '∗' if no information is available.

–t *type* Report on filesystems of a given type (for example, nfs or ufs).

EXAMPLES

A sample of output for **df** looks like:

```
example% df
Filesystem   kbytes   used   avail   capacity   Mounted on
sparky:/      7445     4714  1986      70%      /
sparky:/usr  42277    35291  2758      93%      /usr
```

Note: used+avail is less than the amount of space in the file system (kbytes); this is because the system reserves a fraction of the space in the file system to allow its file system allocation routines to work well. The amount reserved is typically about 10%; this may be adjusted using **tunefs.** When all the space on a file system except for this reserve is in use, only the super-user can allocate new files and data blocks to existing files. When a file system is overallocated in this way, **df** may report that the file system is more than 100% utilized.

FILES

/etc/mnttab	list of file systems currently mounted
/etc/vfstab	list of default parameters for each file system

SEE ALSO

du(1M), **quot**(1M), **tunefs**(1M), **mnttab**(4)

NAME

diff − display line-by-line differences between pairs of text files

SYNOPSIS

diff [**−bitw**] [**−c** | **−e** | **−f** | **−h** | **−n**] *filename1 filename2*
diff [**−bitw**] [**−C** *number*] *filename1 filename2*
diff [**−bitw**] [**−D** *string*] *filename1 filename2*
diff [**−bitw**] [**−c** | **−e** | **−f** | **−h** | **−n**] [**−l**] [**−r**] [**−s**] [**−S** *name*] *directory1 directory2*

AVAILABILITY

SUNWdoc

DESCRIPTION

diff tells what lines must be changed in two files to bring them into agreement. If *filename1* (*filename2*) is −, the standard input is used. If *filename1* (*filename2*) is a directory, then a file in that directory with the name *filename2* (*filename1*) is used. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble **ed** commands to convert *filename1* into *filename2*. The numbers after the letters pertain to *filename2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *filename2* into *filename1*. As in **ed**, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by **<**, then all the lines that are affected in the second file flagged by **>**.

OPTIONS

−b	Ignores trailing blanks (spaces and tabs) and treats other strings of blanks as equivalent.
−i	Ignores the case of letters; for example, ‘**A**’ will compare equal to ‘**a**’.
−t	Expands TAB characters in output lines. Normal or **−c** output adds character(s) to the front of each line that may adversely affect the indentation of the original source lines and make the output lines difficult to interpret. This option will preserve the original source’s indentation.
−w	Ignores all blanks (SPACE and TAB characters) and treats all other strings of blanks as equivalent; for example, ‘**if (a = = b)**’ will compare equal to ‘**if(a= =b)**’.

The following options are mutually exclusive:

−c	Produces a listing of differences with three lines of context. With this option output format is modified slightly: output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen ∗’s. The lines removed from *filename1* are marked with ’—’; those added to *filename2* are marked ’ + ’. Lines that are changed from one file to the other are marked in both files with ’ ! ’.

–C *number*	Produces a listing of differences identical to that produced by **–c** with *number* lines of context.
–e	Produces a script of *a, c*, and *d* commands for the editor **ed**, which will recreate *filename2* from *filename1*. In connection with **–e ,** the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version **ed** scripts ($2,$3,...) made by **diff** need be on hand. A ‘‘latest version’’ appears on the standard output. **(shift; cat $*; echo ′1,$p′) | ed – $1**

Except in rare circumstances, **diff** finds a smallest sufficient set of file differences.

–f	Produces a similar script, not useful with **ed ,** in the opposite order.
–h	Does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **–e** and **–f** are unavailable with **–h .**
–n	Produces a script similar to **–e ,** but in the opposite order and with a count of changed lines on each insert or delete command.
–D *string*	Creates a merged version of *filename1* and *filename2* with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *filename1*, while defining *string* will yield *filename2*.

The following options are used for comparing directories:

–l	Produce output in long format. Before the **diff**, each text file is piped through **pr**(1) to paginate it. Other differences are remembered and summarized after all text file differences are reported.
–r	Applies **diff** recursively to common subdirectories encountered.
–s	Reports files that are the identical; these would not otherwise be mentioned.
–S *name*	Starts a directory **diff** in the middle, beginning with the file *name*.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **diff** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **diff** behaves.

LC_CTYPE

Determines how **diff** handles characters. When **LC_CTYPE** is set to a valid value, **diff** can display and handle text and filenames containing valid characters for that locale. **diff** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **diff** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/tmp/d?????
/usr/lib/diffh for –h
/usr/bin/pr

SEE ALSO

bdiff(1), **cmp**(1), **comm**(1), **ed**(1), **pr**(1), **sdiff**(1), **environ**(5)

DIAGNOSTICS

Exit status is **0** for no differences, **1** for some differences, **2** for trouble.

NOTES

Editing scripts produced under the –**e** or –**f** option are naive about creating lines consisting of a single period (**.**).

Missing newline at end of file indicates that the last line of the file in question did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [–**exEX3**] *filename1 filename2 filename3*

AVAILABILITY

SUNWesu

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*filename1* is different
====2	*filename2* is different
====3	*filename3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f **:** *n1* **a**	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f **:** *n1* **,** *n2* **c**	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n*2, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

The following command will apply the resulting script to *filename1*.

```
(cat script; echo ′1,$p′) | ed – filename1
```

OPTIONS

–e	Produce a script for the editor **ed**(1) that will incorporate into *filename1* all changes between *filename2* and *filename3*, i.e., the changes that normally would be flagged **====** and **====3**.
–x	Produce a script to incorporate only changes flagged **====**.
–3	Produce a script to incorporate only changes flagged **====3**.
–E	Produce a script that will incorporate all changes between *filename2* and *filename3*, but treat overlapping changes (that is, changes that would be flagged with **====** in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by **<<<<<<** and **>>>>>>** lines.
–X	Produce a script that will incorporate only changes flagged **====**, but treat these changes in the manner of the –**E** option.

FILES

/tmp/d3∗
/usr/lib/diff3prog

SEE ALSO

diff(1)

NOTES

Text lines that consist of a single **'.'** will defeat –**e**.

Files longer than 64 Kbytes will not work.

NAME

diffmk – mark differences between versions of a troff input file

SYNOPSIS

diffmk *oldfile newfile markedfile*

AVAILABILITY

SUNWdoc

DESCRIPTION

diffmk compares two versions of a file and creates a third version that includes “change mark” (**.mc**) commands for **nroff**(1) and **troff**(1). *oldfile* and *newfile* are the old and new versions of the file. **diffmk** generates *markedfile*, which, contains the text from *newfile* with **troff**(1) “change mark” requests (**.mc**) inserted where *newfile* differs from *oldfile*. When *markedfile* is formatted, changed or inserted text is shown by **|** at the right margin of each line. The position of deleted text is shown by a single ∗.

EXAMPLES

diffmk can also be used in conjunction with the proper **troff** requests to produce program listings with marked changes. In the following command line:

```
example% diffmk old.c new.c marked.c ; nroff reqs marked.c | pr
```

the file **reqs** contains the following **troff** requests:

```
.pl 1
.ll 77
.nf
.eo
.nh
```

which eliminate page breaks, adjust the line length, set no-fill mode, ignore escape characters, and turn off hyphenation, respectively.

If the characters **|** and ∗ are inappropriate, you might run *markedfile* through **sed**(1) to globally change them.

SEE ALSO

diff(1), **nroff**(1), **sed**(1), **troff**(1)

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, that is, replacing **.sp** by **.sp 2** will produce a “change mark” on the preceding or following line of output.

NAME

dircmp – directory comparison

SYNOPSIS

dircmp [–**d**] [–**s**] [–**w***n*] *dir1 dir2*

AVAILABILITY

SUNWesu

DESCRIPTION

dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

OPTIONS

–**d**	Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in **diff**(1).
–**s**	Suppress messages about identical files.
–**w***n*	Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), **diff**(1)

NAME

dis – object code disassembler

SYNOPSIS

dis [–**C**] [[–**o**] [–**V**] [–**d** *sec*] [–**D** *sec*] [–**F** *function*] [–**l** *string*] [–**t** *sec*] *filename* . . .

DESCRIPTION

The **dis** command produces an assembly language listing of *file*, which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

OPTIONS

The following *options* are interpreted by the disassembler and may be specified in any order.

–**C**	Display demangled C++ symbol names in the disassembly.
–**o**	Print numbers in octal. The default is hexadecimal.
–**V**	Print, on standard error, the version number of the disassembler being executed.
–**d** *sec*	Disassemble the named section as data, printing the offset of the data from the beginning of the section.
–**D** *sec*	Disassemble the named section as data, printing the actual address of the data.
–**F** *function*	Disassemble only the named function in each object file specified on the command line. The –**F** option may be specified multiple times on the command line.
–**l** *string*	Disassemble the archive file specified by *string*. For example, one would issue the command **dis** –**l x** –**l z** to disassemble **libx.a** and **libz.a**, which are assumed to be in **LIBDIR .**
–**t** *sec*	Disassemble the named section as text.

If the –**d**, –**D** or –**t** options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as **[5]**, indicates that the break-pointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information, for example, the –**g** option of **cc**(1B). An expression such as **<40>** in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A function name will appear in the first column, followed by **( )** if the object file contains a symbol table.

ENVIRONMENT

LIBDIR	If this environment variable contains a value, use this as the path to search for the library. If the variable contains a null value, or is not set, it defaults to searching for the library under **/usr/ccs/lib.**

FILES

/usr/ccs/lib	default **LIBDIR**

SEE ALSO

as(1), **cc**(1B), **ld**(1), **a.out**(4)

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NAME

dispgid – displays a list of all valid group names

SYNOPSIS

dispgid

AVAILABILITY

SUNWcsu

DESCRIPTION

dispgid displays a list of all group names on the system (one group per line).

EXIT CODES

0	Successful execution
1	Cannot read the group file

NAME

dispuid – displays a list of all valid user names

SYNOPSIS

dispuid

AVAILABILITY

SUNWcsu

DESCRIPTION

dispuid displays a list of all user names on the system (one line per name).

EXIT CODES

0	Successful execution
1	Cannot read the password file

NAME

dos2unix – convert text file from DOS format to ISO format

SYNOPSIS

dos2unix [**–ascii**] [**–iso**] [**–7**] *originalfile convertedfile*

AVAILABILITY

SUNWesu

DESCRIPTION

dos2unix converts characters in the DOS extended character set to the corresponding ISO standard characters.

This command can be invoked from either DOS or SunOS. However, the filenames must conform to the conventions of the environment in which the command is invoked.

If the original file and the converted file are the same, **dos2unix** will rewrite the original file after converting it.

OPTIONS

–ascii Removes extra carriage returns and converts end of file characters in DOS format text files to conform to SunOS requirements.

–iso This is the default. It converts characters in the DOS extended character set to the corresponding ISO standard characters.

–7 Convert 8 bit DOS graphics characters to 7 bit space characters so that SunOS can read the file.

SEE ALSO

unix2dos(1)

DIAGNOSTICS

File *filename* **not found, or no read permission**

The input file you specified does not exist, or you do not have read permission (check with the SunOS **ls –l** command).

Bad output filename *filename***, or no write permission**

The output file you specified is either invalid, or you do not have write permission for that file or the directory that contains it. Check also that the drive or diskette is not write-protected.

Error while writing to temporary file

An error occurred while converting your file, possibly because there is not enough space on the current drive. Check the amount of space on the current drive using the DIR command. Also be certain that the default diskette or drive is write-enabled (not write-protected). Note that when this error occurs, the original file remains intact.

Could not rename temporary file to
Translated temporary file name = *filename***.**

The program could not perform the final step in converting your file. Your converted file is stored under the name indicated on the second line of this message.

NAME

download – host resident PostScript font downloader

SYNOPSIS

download [–**f**] [–**p** *printer*] [–**m** *name*] [–**H** *directory*] [*file*. . .]

/usr/lib/lp/postscript/download

DESCRIPTION

download prepends host resident fonts to *files* and writes the results on the standard output. If no *files* are specified, or if – is one of the input *files*, the standard input is read. *download* assumes the input *files* make up a single PostScript job and that requested fonts can be included at the start of each input *file.*

Requested fonts are named in a comment (marked with **%%DocumentFonts:**) in the input *files*. Available fonts are the ones listed in the map table selected using the –**m** option.

The map table consists of fontname–file pairs. The fontname is the full name of the PostScript font, exactly as it would appear in a **%%DocumentFonts:** comment. The file is the pathname of the host resident font. A file that begins with a **/** is used as is. Otherwise the pathname is relative to the host font directory. Comments are introduced by % (as in PostScript) and extend to the end of the line.

The only candidates for downloading are fonts listed in the map table that point **download** to readable files. A font is downloaded once, at most. Requests for unlisted fonts or inaccessible files are ignored. All requests are ignored if the map table can not be read.

OPTIONS

–**f**	Force a complete scan of each input *file.* In the absence of an explicit comment pointing *download* to the end of the file, the default scan stops immediately after the PostScript header comments.
–**p** *printer*	Check the list of printer-resident fonts in **/etc/lp/printers/***printer***/residentfonts** before downloading.
–**m** *name*	Use *name* as the font map table. A *name* that begins with **/** is the full pathname of the map table and is used as is. Otherwise *name* is appended to the pathname of the host font directory.
–**H** *directory*	Use *dir* as the host font directory. The default is **/usr/lib/lp/postscript**.

EXAMPLES

The following map table could be used to control the downloading of the Bookman font family:

```
%
% The first string is the full PostScript font name. The second string
% is the file name - relative to the host font directory unless it begins
% with a /.
%

 Bookman-Light          bookman/light
 Bookman-LightItalic    bookman/lightitalic
 Bookman-Demi           bookman/demi
```

Bookman-DemiItalic	**bookman/demiitalic**

Using the file **myprinter/map** (in the default host font directory) as the map table, you could download fonts by issuing the following command:

```
example% download –m myprinter/map file
```

SEE ALSO

dpost(1), **postdaisy**(1), **postdmd**(1), **postio**(1), **postmd**(1), **postprint**(1), **posttek**(1)

DIAGNOSTICS

An exit status of **0** is returned if *files* were successfully processed.

NOTES

The **download** program should be part of a more general program.

download does not look for **%%PageFonts:** comments and there is no way to force multiple downloads of a particular font.

We do not recommend the use of full pathnames in either map tables or the names of map tables.

NAME

dpost – troff postprocessor for PostScript printers

SYNOPSIS

dpost [–**c** *num*] [–**e** *num*] [–**m** *num*] [–**n** *num*] [–**o** *list*] [–**w** *num*] [–**x** *num*] [–**y** *num*]
[–**F** *dir*] [–**H** *dir*] [–**L** *file*] [–**O**] [–**T** *name*] [*file*. . .]

/usr/lib/lp/postscript/dpost

DESCRIPTION

dpost translates *files* created by **troff**(1) into PostScript and writes the results on the standard output. If no *files* are specified, or if – is one of the input *files*, the standard input is read.

The *files* should be prepared by **troff**. The default font files in **/usr/lib/font/devpost** produce the best and most efficient output. They assume a resolution of 720 dpi, and can be used to format files by adding the –**Tpost** option to the **troff** call. Older versions of the **eqn** and **pic** preprocessors need to know the resolution that **troff** will be using to format the *files*. If those are the versions installed on your system, use the –**r720** option with **eqn** and –**T720** with **pic**.

dpost makes no assumptions about resolutions. The first **x res** command sets the resolution used to translate the input *files*, the **DESC.out** file, usually **/usr/lib/font/devpost/DESC.out**, defines the resolution used in the binary font files, and the PostScript prologue is responsible for setting up an appropriate user coordinate system.

OPTIONS

–**c** *num*	Print *num* copies of each page. By default only one copy is printed.
–**e** *num*	Sets the text encoding level to *num*. The recognized choices are 0, 1, and 2. The size of the output file and print time should decrease as *num* increases. Level 2 encoding will typically be about 20 percent faster than level 0, which is the default and produces output essentially identical to previous versions of **dpost .**
–**m** *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, which is located near the upper left corner of each page. The default magnification is **1.0.**
–**n** *num*	Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default, *num* is set to **1.**
–**o** *list*	Print those pages for which numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1*–*N2.* A missing *N1* means the lowest numbered page, a missing *N2* means the highest. The page range is an expression of logical pages rather than physical sheets of paper. For example, if you are printing two logical pages to a sheet, and you specified a range of **4,** then two sheets of paper would print, containing four page layouts. If you specified a page range of **3-4,** when requesting two logical pages to a sheet; then *only* page 3 and page 4 layouts would print, and they would appear on one physical sheet of paper.

–p *mode*	Print *files* in either portrait or landscape *mode.* Only the first character of *mode* is significant. The default *mode* is portrait.
–w *num*	Set the line width used to implement *troff* graphics commands to *num* points, where a point is approximately 1/72 of an inch. By default, *num* is set to **0.3** points.
–x *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed near the upper left corner of the page, with positive x to the right and positive y down the page. Positive *num* moves everything right. The default offset is **0** inches.
–y *num*	Translate the origin *num* inches along the positive y axis. Positive *num* moves text up the page. The default offset is **0.**
–F *dir*	Use *dir* as the font directory. The default *dir* is **/usr/lib/font**, and **dpost** reads binary font files from directory **/usr/lib/font/devpost**.
–H *dir*	Use *dir* as the host resident font directory. Files in this directory should be complete PostScript font descriptions, and must be assigned a name that corresponds to the appropriate two-character **troff** font name. Each font file is copied to the output file only when needed and at most once during each job. There is no default directory.
–L *file*	Use *file* as the PostScript prologue which, by default, is **/usr/lib/lp/postscript/dpost.ps**.
–O	Disables PostScript picture inclusion. A recommended option when **dpost** is run by a spooler in a networked environment.
–T *name*	Use font files for device *name* as the best description of available PostScript fonts. By default, *name* is set to **post** and **dpost** reads binary files from **/usr/lib/font/devpost**.

EXAMPLES

If the old versions of **eqn** and **pic** are installed on your system, you can obtain the best possible looking output by issuing a command line such as the following:

```
example% pic –T720 file | tbl | eqn –r720 | troff –mm –Tpost | dpost
```

Otherwise,

```
example% pic file | tbl | eqn | troff –mm –Tpost | dpost
```

should give the best results.

FILES

/usr/lib/font/devpost/*.out
/usr/lib/font/devpost/charlib/*
/usr/lib/lp/postscript/color.ps
/usr/lib/lp/postscript/draw.ps
/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/ps.requests
/usr/lib/macros/pictures
/usr/lib/macros/color

SEE ALSO

download(1), **postdaisy**(1), **postdmd**(1), **postio**(1), **postmd**(1), **postprint**(1), **postreverse**(1), **posttek**(1), **troff**(1)

DIAGNOSTICS

An exit status of **0** is returned if *files* have been translated successfully, while 2 often indicates a syntax error in the input *files*.

NOTES

Output files often do not conform to Adobe's file structuring conventions. Piping the output of **dpost** through **postreverse**(1) should produce a minimally conforming PostScript file.

Although **dpost** can handle files formatted for any device, emulation is expensive and can easily double the print time and the size of the output file. No attempt has been made to implement the character sets or fonts available on all devices supported by **troff**. Missing characters will be replaced by white space, and unrecognized fonts will usually default to one of the Times fonts (that is, **R**, **I**, **B**, or **BI**).

An **x res** command must precede the first **x init** command, and all the input *files* should have been prepared for the same output device.

Use of the **−T** option is not encouraged. Its only purpose is to enable the use of other PostScript font and device description files, that perhaps use different resolutions, character sets, or fonts.

Although level 0 encoding is the only scheme that has been thoroughly tested, level 2 is fast and may be worth a try.

NAME

du – display the number of disk blocks used per directory or file

SYNOPSIS

/usr/ucb/du

/usr/ucb/du [**–a**] [**–s**] [*filename* **]**

AVAILABILITY

SUNWscpu

DESCRIPTION

du gives the number of kilobytes contained in all files and, recursively, directories within each specified directory or file *filename*. If *filename* is missing, '**.**' (the current directory) is used.

A file which has multiple links to it is only counted once.

OPTIONS

–a Generate an entry for each file.

–s Only display the grand total for each of the specified *filename*s.

Entries are generated only for each directory in the absence of options.

EXAMPLES

Here is an example of using **du** in a directory. We used the **pwd**(1) command to identify the directory, then used **du** to show the usage of all the subdirectories in that directory. The grand total for the directory is the last entry in the display:

```
example% pwd
/usr/ralph/misc
example% du
5       ./jokes
33      ./squash
44      ./tech.papers/lpr.document
217     ./tech.papers/new.manager
401     ./tech.papers
144     ./memos
80      ./letters
388     ./window
93      ./messages
15      ./useful.news
1211    .
example%
```

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **du** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **du** behaves.

LC_CTYPE

Determines how **du** handles characters. When **LC_CTYPE** is set to a valid value, **du** can display and handle text and filenames containing valid characters for that locale. **du** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **du** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

pwd(1), **df**(1M), **quot**(1M), **environ**(5)

NOTES

Filename arguments that are not directory names are ignored, unless you use **–a**.

If there are too many distinct linked files, **du** will count the excess files more than once.

NAME

dump – dump selected parts of an object file

SYNOPSIS

dump [**–aCcDfghLlorstV**] [**–T** *index* [*, indexn*]] *filename* ...
dump [**–afhorstL** [**v**]] *filename* ...
dump [**–hsr** [**–d** *number* [*, numbern*]]] *filename* ...
dump [**–hsrt** [**–n** *name*]] *filename* ...

DESCRIPTION

The **dump** command dumps selected parts of each of its object *file* arguments.

OPTIONS

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

–a	Dump the archive header of each member of an archive.
–c	Dump the string table(s).
–C	Dump decoded C++ symbol table names.
–D	Dump debugging information.
–f	Dump each file header.
–g	Dump the global symbols in the symbol table of an archive.
–h	Dump the section headers.
–l	Dump line number information.
–L	Dump dynamic linking information and static shared library information, if available.
–o	Dump each program execution header.
–r	Dump relocation information.
–s	Dump section contents in hexadecimal.
–t	Dump symbol table entries.
–T *index* or **–T** *index1***,***index2*	Dump only the indexed symbol table entry defined by *index* or a range of entries defined by *index1***,***index2*.
–V	Print version information.

The following modifiers are used in conjunction with the options listed above to modify their capabilities.

–d *number* or **–d** *number1***,***number2*
Dump the section number indicated by *number* or the range of sections starting at *number1* and ending at *number2* . This modifier can be used with **–h**, **–s**, and **–r**. When **–d** is used with **–h** or **–s**, the argument is treated as the number of a section or range of sections. When **–d** is used with **–r**, the argument is treated as the number of the section or range of sections to which the relocation applies. For example, to print out all relocation entries associated with the **.text** section, specify the number of the section as the argument to **–d**. If **.text** is section number 2 in the file, **dump –r –d 2** will print all

associated entries. To print out a specific relocation section use **dump** –**s** –**n** *name* for raw data output, or **dump** –**sv** –**n** *name* for interpreted output.

–**n** *name* Dump information pertaining only to the named entity. This modifier can be used with –**h**, –**s**, –**r**, and –**t**. When –**n** is used with –**h** or –**s**, the argument will be treated as the name of a section. When –**n** is used with –**t** or –**r**, the argument will be treated as the name of a symbol. For example, **dump** –**t** –**n .text** will dump the symbol table entry associated with the symbol whose name is **dump** –**h** –**n .text** will dump the section header information for the **section.**

–**p** Suppress printing of the headings.

–**v** Dump information in symbolic representation rather than numeric. This modifier can be used with –**a** (date, user id, group id), –**f** (class, data, type, machine, version, flags), –**h** (type, flags), –**o** (type, flags), –**r** (name, type), –**s** (interpret section contents wherever possible), –**t** (type, bind), and –**L** (value). When –**v** is used with –**s**, all sections that can be interpreted, such as the string table or symbol table, will be interpreted. For example, **dump** –**sv** –**n .symtab** *filename*. . . will produce the same formatted output as **dump** –**tv** *filename*. . . , but **dump** –**s** –**n .symtab** *filename*. . . will print raw data in hexadecimal. Without additional modifiers, **dump** –**sv** *filename*. . . will dump all sections in the files interpreting all those that it can and dumping the rest (such as **.text** or **.data**) as raw data.

The **dump** command attempts to format the information it dumps in a meaningful way, printing certain information in character, hexadecimal, octal or decimal representation as appropriate.

SEE ALSO

nm(1), **a.out**(4), **ar**(4)

NAME

dumpcs – show codeset table for the current locale

SYNOPSIS

dumpcs [**–0123vw**]

DESCRIPTION

dumpcs shows a list of printable characters for the user's current locale, along with their hexadecimal code values. The display device is assumed to be capable of displaying characters for a given locale. With no option, **dumpcs** displays the entire list of printable characters for the current locale.

With one or more numeric options specified, it shows EUC codeset(s) for the current locale according to the numbers specified, and in order of codeset number. Each non-printable character is represented by an asterisk ''*'' and enough ASCII space character(s) to fill that codeset's column width.

OPTIONS

–0	Show ASCII (or EUC primary) codeset.
–1	Show EUC codeset 1, if used for the current locale.
–2	Show EUC codeset 2, if used for the current locale.
–3	Show EUC codeset 3, if used for the current locale.
–v	Verbose. Normally, ranges of non-printable characters are collapsed into a single line. This option produces one line for each non-printable character.
–w	Replace code values with corresponding wide character values (process codes).

ENVIRONMENT

The environment variables **LC_CTYPE** and **LANG** control the character classification throughout **dumpcs**. On entry to **dumpcs**, these environment variables are checked in that order. This implies that a new setting for **LANG** does not override the setting of **LC_CTYPE**. When none of the values is valid, the character classification defaults to the POSIX.1 ''C'' locale.

FILES

/usr/lib/locale/*locale-name***/LC_CTYPE/ctype**
data file containing character classification, conversion, and character set width information

SEE ALSO

chrtbl(1M)

NAME

echo – echo arguments

SYNOPSIS

echo [*argument*] . . .
echo [–**n**] [*argument*]

AVAILABILITY

SUNWcsu

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output.

/usr/bin/sh (Bourne shell) understands the following C-like escape conventions, however, beware of conflicts with the shell's use of \ :

Note that the escape conventions are not available if the BSD version of the Bourne shell *echo* command is used. (See OPTIONS section)

\ **b**	backspace
\ **c**	print line without new-line
\ **f**	form-feed
\ **n**	new-line
\ **r**	carriage return
\ **t**	tab
\ **v**	vertical tab
\ \	backslash
\ **0***n*	where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

echo is useful for producing diagnostics in command files, for sending known data into a pipe, and for displaying the contents of environment variables.

OPTIONS

The –**n** option is available to **/usr/bin/sh** users only if **/usr/ucb** precedes **/usr/bin** in the user's PATH, however, then the backslash characters, described above, are NOT available. –**n** is available to **/usr/bin/csh** users, regardless of PATH:

–**n**	Do not add the newline to the output.

SEE ALSO

sh(1), **shell_builtins**(1), **ascii**(5)

NOTES

The –**n** option is a transition aid for BSD applications, and may not be supported in future releases.

When representing an 8-bit character by using the escape convention \ **0***n*, the *n* must **always** be preceded by the digit zero (0).

For example, typing: **echo ´WARNING:\ 07´** will print the phrase **WARNING:** and sound the ''bell'' on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the '' \'' that precedes the ''07''.

Following the **\ 0**, up to three digits are used in constructing the octal output character. If, following the **\ 0***n*, you want to echo additional digits that are not part of the octal representation, you must use the full 3-digit *n*. For example, if you want to echo ''ESC 7'' you must use the three digits ‘‘033’’ rather than just the two digits ‘‘33’’ after the **\ 0**.

```
2 digits    Incorrect:    echo "\0337"  | od -xc
            produces:     df0a                      (hex)
                          337                       (ascii)

3 digits    Correct:      echo "\00337" | od -xc
            produces:     lb37 0a00                 (hex)
                          033 7                     (ascii)
```

For the octal equivalents of each character, see **ascii**(5).

NAME

echo – echo arguments to standard output

SYNOPSIS

/usr/ucb/echo [*argument*] . . .
/usr/ucb/echo [**–n**] [*argument*]

AVAILABILITY

SUNWscpu

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output.

The **/usr/bin/sh** version understands the following C-like escape conventions; beware of conflicts with the shell's use of \\:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\\\	backslash
\0*n*	where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

echo is useful for producing diagnostics in command files and for sending known data into a pipe.

OPTIONS

–n Do not add the newline to the output.

SEE ALSO

sh(1)

NOTES

The **–n** option is a transition aid for BSD applications, and may not be supported in future releases.

When representing an 8-bit character by using the escape convention **\0***n*, the *n* must **always** be preceded by the digit zero (0).

For example, typing: **echo ´WARNING:\07´** will print the phrase **WARNING:** and sound the ‘‘bell’’ on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the ‘‘\’’ that precedes the ‘‘07’’.

For the octal equivalents of each character, see **ascii**(5).

NAME

echo – put string on virtual output

SYNOPSIS

echo [*string* . . .]

DESCRIPTION

The **echo** function directs each string it is passed to the standard output. If no argument is given, **echo** looks to the standard input for input. It is often used in conditional execution or for passing a string to another command.

EXAMPLES

Set the **done** descriptor to **help** if a test fails:

```
done=`if [ -s $F1 ];
        then echo close;
        else echo help;
        fi`
```

SEE ALSO

echo(1)

NAME

ed, red – text editor

SYNOPSIS

ed [– | –**s**] [–**p** *string*] [–**x**] [–**C**] [*filename*]

red [– | –**s**] [–**p** *string*] [–**x**] [–**C**] [*filename*]

AVAILABILITY

SUNWcsu

DESCRIPTION

ed is the standard text editor. If the *filename* argument is given, **ed** simulates an **e** command (see below) on the named file; that is to say, the file is read into **ed**'s buffer so that it can be edited.

ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a **w** (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

red is a restricted version of **ed**. It will only allow editing of files in the current directory. It prohibits executing shell commands via **!***shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both **ed** and **red** support the **fspec**(4) formatting capability. After including a format specification as the first line of *filename* and invoking **ed** with your terminal in **stty** –**tabs** or **stty tab3** mode (see **stty**(1)), the specified tab stops will automatically be used when scanning *filename*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. Note: when you are entering text into the file, this format is not in effect; instead, because of being in **stty** –**tabs** or **stty tab3** mode, tabs are expanded to every eighth column.

Commands to **ed** have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While **ed** is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Leave input mode by typing a period (**.**) at the beginning of a line, followed immediately by a carriage return.

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (for example, **s**) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by **ed** are constructed as follows:

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\ ) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

- a. **.**, *****, **[**, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets (**[ ]**; see 1.4 below).
- b. **^** (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 4.1 and 4.3 below), or when it immediately follows the left of a pair of square brackets (**[ ]**) (see 1.4 below).
- c. **$** (dollar sign), which is special at the **end** of an *entire* RE (see 4.2 below).
- d. The character used to bound (that is, delimit) an entire RE, which is special for that RE (for example, see how slash (**/**) is used in the **g** command, below.)

1.3 A period (**.**) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets (**[ ]**) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (**^**), the one-character RE matches any character *except* new-line and the remaining characters in the string. The **^** has this special meaning *only* if it occurs first in the string. The minus (–) may be used to indicate a range of consecutive characters; for example, **[0–9]** is equivalent to **[0123456789]**. The – loses this special meaning if it occurs first (after an initial **^**, if any) or last in the string. The right square bracket (**]**) does not terminate such a string when it is the first character within it (after an initial **^**, if any); for example, **[ ]a–f]** matches either a right square bracket (**]**) or one of the ASCII letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct RE s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*****) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by **\\{***m***\\}**, **\\{***m,***\\}**, or **\\{***m,n***\\}** is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; **\\{***m***\\}** matches *exactly m* occurrences; **\\{***m,***\\}** matches *at least m* occurrences; **\\{***m,n***\\}** matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences **\\(** and **\\)** is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between **\(** and **\)** *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of **\(** counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

A RE may be constrained to match words.

3.1 **\<** constrains a RE to match the beginning of a string or to follow a character that is not a digit, underscore, or letter. The first character matching the RE must be a digit, underscore, or letter.

3.2 **\>** constrains a RE to match the end of a string or to precede a character that is not a digit, underscore, or letter.

An *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

4.1 A circumflex (**^**) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

4.2 A dollar sign (**$**) at the end of an entire RE constrains that RE to match a *final* segment of a line.

4.3 The construction **^***entire RE* **$** constrains the entire RE to match the entire line.

The null RE (for example, **//**) is equivalent to the last RE encountered. See also the last paragraph before FILES below.

To understand addressing in **ed** it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. ′*x* addresses the line marked with the mark name character *x*, which must be an ASCII lower-case letter (**a**–**z**). Lines are marked with the **k** command described below.
5. A RE enclosed by slashes (**/**) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before FILES below.
6. A RE enclosed in question marks (**?**) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before FILES below.

7. An address followed by a plus sign (**+**) or a minus sign (–) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. A shorthand for .+5 is .5.
8. If an address begins with **+** or –, the addition or subtraction is taken with respect to the current line; for example, –**5** is understood to mean **.**–**5**.
9. If an address ends with **+** or –, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address – refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character **^** in addresses is entirely equivalent to –.) Moreover, trailing **+** and – characters have a cumulative effect, so –– refers to the current line less 2.
10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the first address is calculated, the current line (.) is set to that value, and then the second address is calculated. This feature can be used to determine the starting line for forward and backward searches (see Rules 5 and 6, above). The second address of any two-address sequence must correspond to a line in the buffer that follows the line corresponding to the first address.

In the following list of **ed** commands, the parentheses shown prior to the command are *not* part of the address; rather they show the default address(es) for the command.

It is generally illegal for more than one command to appear on a line. However, any command (except **e**, **f**, **r**, or **w**) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the **l**, **n**, and **p** commands.

(.)a
<text>
.

The **a**ppend command accepts zero or more lines of text and appends it after the addressed line in the buffer. The current line (**.**) is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the ‘‘appended’’ text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The **c**hange command deletes the addressed lines from the buffer, then accepts zero or more lines of text that replaces these lines in the buffer. The current line (**.**) is left at the last line input, or, if there were none, at the first line that was not deleted.

C

Same as the **X** command, described later, except that **ed** assumes all text read in for the **e** and **r** commands is encrypted unless a null key is typed in.

(.,.)d

The **d**elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *filename*

The **e**dit command deletes the entire contents of the buffer and then reads the contents of *filename* into the buffer. The current line (**.**) is set to the last line of the buffer. If *filename* is not given, the currently remembered file name, if any, is used (see the **f** command). The number of characters read in is printed; *filename* is remembered for possible use as a default file name in subsequent **e**, **r**, and **w** commands. If *filename* is replaced by **!**, the rest of the line is taken to be a shell (**sh**(1)) command whose output is to be read in. Such a shell command is *not* remembered as the current file name. See also DIAGNOSTICS below.

E *filename*

The **E**dit command is like **e**, except that the editor does not check to see if any changes have been made to the buffer since the last **w** command.

f *filename*

If *filename* is given, the **f** ile-name command changes the currently remembered file name to *filename*; otherwise, it prints the currently remembered file name.

(1,$)g/*RE/command list*

In the **g**lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with the current line (**.**) initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; **a**, **i**, and **c** commands and associated input are permitted. The **.** terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the **p** command. The **g**, **G**, **v**, and **V** commands are *not* permitted in the *command list*. See also the NOTES and the last paragraph before FILES below.

(1,$)G/*RE***/**

In the interactive **G**lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, the current line (**.**) is changed to that line, and any *one* command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of **G**. Note: The commands input as part of the execution of the **G** command may address and affect *any* lines in the buffer. The **G** command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The **h**elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The **H**elp command causes **ed** to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The **H** command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The **i**nsert command accepts zero or more lines of text and inserts it before the addressed line in the buffer. The current line (**.**) is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

The **j**oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar**k** command marks the addressed line with name *x*, which must be an ASCII lower-case letter (**a**–**z**). The address ′*x* then addresses this line; the current line (**.**) is unchanged.

(.,.)l

The **l**ist command prints the addressed lines in an unambiguous way: a few non-printing characters (for example, *tab, backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An **l** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)m*a*

The **m**ove command repositions the addressed line(s) after the line addressed by *a*. Address **0** is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; the current line (**.**) is left at the last line moved.

(.,.)n

The **n**umber command prints the addressed lines, preceding each line by its line number and a tab character; the current line (**.**) is left at the last line printed. The **n** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)p

The **p**rint command prints the addressed lines; the current line (**.**) is left at the last line printed. The **p** command may be appended to any command other than **e**, **f**, **r**, or **w**. For example, **dp** deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The **P** command alternately turns this mode on and off; it is initially off.

q

The **q**uit command causes **ed** to exit. No automatic write of a file is done; however, see DIAGNOSTICS , below.

Q

The editor exits without checking if changes have been made in the buffer since the last **w** command.

($)r *filename*

The **r**ead command reads the contents of *filename* into the buffer. If *filename* is not given, the currently remembered file name, if any, is used (see the **e** and **f** commands). The currently remembered file name is *not* changed unless *filename* is the very first file name mentioned since **ed** was invoked. Address 0 is legal for **r** and causes the file to be read in at the beginning of the buffer. If the read is successful, the number of characters read in is printed; the current line (**.**) is set to the last line read in. If *filename* is replaced by **!**, the rest of the line is taken to be a shell (see **sh**(1)) command whose output is to be read in. For example, **$r !ls** appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE***/***replacement***/** or
(.,.)s/*RE***/***replacement***/g** or
(.,.)s/*RE***/***replacement***/***n* *n* = 1-512

The **s**ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number *n*, appears after the command, only the *n*-th occurrence of the matched string on each addressed line is replaced. It is an error if the substitution fails on *all* addressed lines. Any character other than space or new-line may be used instead of **/** to delimit the RE and the *replacement*; the current line (**.**) is left at the last line on which a substitution occurred. See also the last paragraph before FILES below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by \. As a more general feature, the characters **\***n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between **\\(** and **\\)**. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of **\\(** starting from the left. When the character **%** is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The **%** loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a **g** or **v** command list.

(.,.)t*a*

This command acts just like the **m** command, except that a *copy* of the addressed lines is placed after address **a** (which may be 0); the current line (**.**) is left at the last line copied.

u

The **u**ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, **G**, or **V** command.

(1,$)v/*RE***/***command list*

This command is the same as the global command **g**, except that the lines marked during the first step are those that do *not* match the RE.

(1,$)V/*RE***/**

This command is the same as the interactive global command **G**, except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *filename*

The **w**rite command writes the addressed lines into *filename*. If *filename* does not exist, it is created with mode **666** (readable and writable by everyone), unless your file creation mask dictates otherwise; see the description of the **umask** special command on **sh**(1). The currently remembered file name is *not* changed unless *filename* is the very first file name mentioned since **ed** was invoked. If no file name is given, the currently remembered file name, if any, is used (see the **e** and **f** commands); the current line (**.**) is unchanged. If the command is successful, the number of characters written is printed. If *filename* is replaced by **!**, the rest of the line is taken to be a shell (see **sh**(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

(1,$)W *filename*

This command is the same as the **w**rite command above, except that it appends the addressed lines to the end of *filename* if it exists. If *filename* does not exist, it is created as described above for the *w* command.

X

An educated guess is made to determine whether text read in for the **e** and **r** commands is encrypted. A null key turns off encryption. Subsequent **e**, **r**, and **w** commands will use this key to encrypt or decrypt the text. An explicitly empty key turns off encryption. Also, see the –**x** option of **ed**.

($)=

The line number of the addressed line is typed; the current line (**.**) is unchanged by this command.

!*shell command*

The remainder of the line after the **!** is sent to the UNIX system shell (see **sh**(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; the current line (**.**) is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, **ed** prints a **?** and returns to *its* command level.

Some size limitations: 512 characters in a line, 256 characters in a global command list, and 255 characters in the pathname of a file (counting slashes). The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters.

If a file is not terminated by a new-line character, **ed** adds one and puts out a message explaining what it did.

If the closing delimiter of a RE or of a replacement string (for example, **/**) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	**s/s1/s2/p**
g/s1	**g/s1/p**
?s1	**?s1?**

OPTIONS

–C	Encryption option; the same as the **–x** option, except that **ed** simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.
–p	Allows the user to specify a prompt string.
–s	Suppresses the printing of character counts by **e**, **r**, and **w** commands, of diagnostics from **e** and **q** commands, and of the **!** prompt after a **!***shell command*.
–x	Encryption option; when used, **ed** simulates an **X** command and prompts the user for a key. The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **–x** option. Also, see the NOTES section at the end of this manual page.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **ed** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **ed** behaves.

LC_CTYPE

Determines how **ed** handles characters. When **LC_CTYPE** is set to a valid value, **ed** can display and handle text and filenames containing valid characters for that locale. **ed** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **ed** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

$TMPDIR	If this environment variable is not null, its value is used in place of **/var/tmp** as the directory name for the temporary work file.
/var/tmp	If **/var/tmp** exists, it is used as the directory name for the temporary work file.
/tmp	If the environment variable **TMPDIR** does not exist or is null, and if **/var/tmp** does not exist, then **/tmp** is used as the directory name for the temporary work file.
ed.hup	Work is saved here if the terminal is hung up.

SEE ALSO

edit(1), **ex**(1), **grep**(1), **sed**(1), **sh**(1), **stty**(1), **umask**(1), **vi**(1), **fspec**(4), **environ**(5), **regexp**(5)

DIAGNOSTICS

?	for command errors.
?*filename*	for an inaccessible file. (use the **h**elp and **H**elp commands for detailed explanations).

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, **ed** warns the user if an attempt is made to destroy **ed**'s buffer via the **e** or **q** commands. It prints **?** and allows one to continue editing. A second **e** or **q** command at this point will take effect. The –**s** command-line option inhibits this feature.

NOTES

The – option, although it continues to be supported, has been replaced in the documentation by the –**s** option that follows the Command Syntax Standard (see **intro**(1)).

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

A **!** command cannot be subject to a **g** or a **v** command.

The **!** command and the **!** escape from the **e**, **r**, and **w** commands cannot be used if the editor is invoked from a restricted shell (see **sh**(1)).

The sequence **\n** in a RE does not match a new-line character.

If the editor input is coming from a command file (for example, **ed** *filename* **<** *ed_cmd_file*), the editor exits at the first failure.

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [– | –**s**] [–**l**] [–**L**] [–**R**] [–**r** [*filename*]] [–**t** *tag*] [–**v**] [–**V**] [–**x**] [–**w***n*] [–**C**]
[+*command* | –**c** *command*] *filename*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

edit is a variant of the text editor **ex** recommended for new or casual users who wish to use a command-oriented editor. It operates precisely as **ex** with the following options automatically set:

novice	**ON**
report	**ON**
showmode	**ON**
magic	**OFF**

The following brief introduction should help you get started with **edit**. If you are using a CRT terminal you may want to learn about the display editor **vi**.

To edit the contents of an existing file you begin with the command **edit** *name* to the shell. **edit** makes a copy of the file that you can then edit, and tells you how many lines and characters are in the file. To create a new file, you also begin with the command **edit** with a filename: **edit** *name*; the editor will tell you it is a **[New File]**.

The **edit** command prompt is the colon (**:**), which you should see after starting the editor. If you are editing an existing file, then you will have some lines in **edit**'s buffer (its name for the copy of the file you are editing). When you start editing, **edit** makes the last line of the file the current line. Most commands to **edit** use the current line if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and type carriage return (as you should after all **edit** commands), the current line will be printed. If you **delete** (**d**) the current line, **edit** will print the new current line, which is usually the next line in the file. If you **delete** the last line, then the new last line becomes the current one.

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you execute this command (typing a carriage return after the word **append**), **edit** will read lines from your terminal until you type a line consisting of just a dot (**.**); it places these lines after the current line. The last line you type then becomes the current line. The **insert** (**i**) command is like **append**, but places the lines you type before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you execute the command **1**, then **edit** will type the first line of the buffer. If you then execute the command **d**, **edit** will delete the first line, line 2 will become line 1, and **edit** will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command: **s**/*old* /*new*/ where *old* is the string of characters you want to replace and *new* is the string of characters you want to replace *old* with.

The **filename** (**f**) command will tell you how many lines there are in the buffer you are editing and will say **[Modified]** if you have changed the buffer. After modifying a file, you can save the contents of the file by executing a **write** (**w**) command. You can leave the editor by issuing a **quit** (**q**) command. If you run **edit** on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from **edit** after modifying the buffer without writing it out, you will receive the message **No write since last change (:quit! overrides)**, and **edit** will wait for another command. If you do not want to write the buffer out, issue the **quit** command followed by an exclamation point (**q!**). The buffer is then irretrievably discarded and you return to the shell.

By using the **d** and **a** commands and giving line numbers to see lines in the file, you can make any changes you want. You should learn at least a few more things, however, if you will use **edit** more than a few times.

The **change** (**c**) command changes the current line to a sequence of lines you supply (as in **append**, you type lines up to a line consisting of only a dot (**.**). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, that is, **3,5c**. You can print lines this way too: **1,23p** prints the first 23 lines of the file.

The **undo** (**u**) command reverses the effect of the last command you executed that changed the buffer. Thus if you execute a **substitute** command that does not do what you want, type **u** and the old contents of the line will be restored. You can also **undo** an **undo** command. **edit** will give you a warning message when a command affects more than one line of the buffer. Note that commands such as **write** and **quit** cannot be undone.

To look at the next line in the buffer, type carriage return. To look at a number of lines, type **ˆD** (while holding down the control key, press **d**) rather than carriage return. This will show you a half-screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at nearby text by executing the **z** command. The current line will appear in the middle of the text displayed, and the last line displayed will become the current line; you can get back to the line where you were before you executed the **z** command by typing ´´. The **z** command has other options: **z–** prints a screen of text (or 24 lines) ending where you are; **z+** prints the next screenful. If you want less than a screenful of lines, type **z.11** to display five lines before and five lines after the current line. (Typing **z.***n*, when *n* is an odd number, displays a total of *n* lines, centered about the current line; when *n* is an even number, it displays *n*–1 lines, so that the lines displayed are centered around the current line.) You can give counts after other commands; for example, you can delete 5 lines starting with the current line with the command **d5.**

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form /*text*/ to search forward for *text* or ?*text*? to search backward for *text* . If a search reaches the end of the file without finding *text*, it wraps around and continues to search back to the line where you are. A useful feature here is a search of the form /^*text*/ which searches for *text* at the beginning of a line. Similarly /*text*$/ searches for *text* at the end of a line. You can leave off the trailing **/** or **?** in these commands.

The current line has the symbolic name dot (**.**); this is most useful in a range of lines as in **.,$p** which prints the current line plus the rest of the lines in the file. To move to the last line in the file, you can refer to it by its symbolic name $. Thus the command **$d** deletes the last line in the file, no matter what the current line is. Arithmetic with line references is also possible. Thus the line **$–5** is the fifth before the last and **.+20** is 20 lines after the current line.

You can find out the current line by typing **'.='**. This is useful if you wish to move or copy a section of text within a file or between files. Find the first and last line numbers you wish to copy or move. To move lines 10 through 20, type **10,20d a** to delete these lines from the file and place them in a buffer named **a**. **edit** has 26 such buffers named **a** through **z**. To put the contents of buffer **a** after the current line, type **put a**. If you want to move or copy these lines to another file, execute an **edit** (**e**) command after copying the lines; following the **e** command with the name of the other file you wish to edit, that is, **edit chapter2**. To copy lines without deleting them, use **yank** (**y**) in place of **d**. If the text you wish to move or copy is all within one file, it is not necessary to use named buffers. For example, to move lines 10 through 20 to the end of the file, type **10,20m $**.

OPTIONS

These options can be turned on or off using the **set** command in **ex**(1).

– | **–s**	Suppress all interactive user feedback. This is useful when processing editor scripts.
–l	Set up for editing LISP programs.
–L	List the name of all files saved as the result of an editor or system crash.
–R	**Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.
–r *filename*	Edit *filename* after an editor or system crash. (Recovers the version of *filename* that was in the buffer when the crash occurred.)
–t *tag*	Edit the file containing the *tag* and position the editor at its definition.
–v	Start up in display editing state using **vi**. You can achieve the same effect by simply typing the **vi** command itself.
–V	Verbose. Any non-tty input will be echoed on standard error. This may be useful when processing editor commands within shell scripts.
–x	Encryption option; when used, **edit** simulates the **X** command of **ex** and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of the **crypt** command. The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **–x** option.
–w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.

−C	Encryption option; same as the **−x** option, except that **vi** simulates the **C** command of **ex**. The **C** command is like the **X** command of **ex**, except that all text read in is assumed to have been encrypted.
+*command* | **−c** *command*	Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *filename* argument indicates one or more files to be edited.

SEE ALSO

ed(1), **ex**(1), **vi**(1)

NOTES

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

NAME

egrep – search a file for a pattern using full regular expressions

SYNOPSIS

egrep [**–bchilnsv**] [**–e** *special-expression*] [**–f** *filename*] [*strings*] [*filename* ...]

AVAILABILITY

SUNWcsu

DESCRIPTION

egrep (*expression grep*) searches files for a pattern of characters and prints all lines that contain that pattern. **egrep** uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

egrep accepts full regular expressions as in **ed**(1), except for **\(** and **\)**, with the addition of:

1. A full regular expression followed by **+** that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by **?** that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses **( )** for grouping.

Be careful using the characters **$**, ∗, **[**, **^**, | , **(**, **)**, and \ in *full regular expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes ′...′.

The order of precedence of operators is **[ ]**, then ∗ **? +**, then concatenation, then | and new-line.

If no files are specified, **egrep** assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

OPTIONS

–b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

–c Print only a count of the lines that contain the pattern.

–h Suppress printing of filenames when searching multiple files.

–i Ignore upper/lower case distinction during comparisons.

–l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

–n Precede each line by its line number in the file (first line is 1).

–s Work silently, that is, display nothing except error messages. This is useful for checking the error status.

–v Print all lines except those that contain the pattern.

–e *special-expression*
Search for a *special expression* (*full regular expression* that begins with a –).

–f *filename* Take the list of *full regular expressions* from *filename*.

SEE ALSO

ed(1), **fgrep**(1), **grep**(1), **sed**(1), **sh**(1)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

NOTES

Ideally there should be only one **grep** command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **<stdio.h>.**

NAME

eject – eject media such as CD-ROM and floppy from drive

SYNOPSIS

eject [**−dfnq**] [*device* | *nickname*]

AVAILABILITY

SUNWcsu

DESCRIPTION

eject is used for those removable media devices that do not have a manual eject button, or for those that do, but are managed by Volume Management. The device may be specified by its name or by a nickname; if no device is specified the default device is used.

Only devices that support **eject** under program control respond to this command. When **eject** is used on media that can only be ejected manually, it will do everything except remove the media — including unmounting the file system if it is mounted (see **rmmount**(1M)). In this case, **eject** displays a message that the media can now be manually ejected. If a window system is running, the message is displayed as a pop-up window. If no window system is running, a message is displayed both to **stderr** and to the system console that the media can now be physically removed.

It is not recommended to physically eject media from a device which contains mounted filesystems. **eject** automatically searches for any mounted filesystems which reside on the device and attempts to **umount** them prior to ejecting the media (see **mount**(1M)). If the unmount operation fails, **eject** prints a warning message and exits. The **−f** flag may be used to specify an eject *even* if the device contains mounted partitions.

eject can also display its default device and a list of nicknames.

If you have inserted a floppy diskette, you must use **volcheck**(1) before ejecting the media to inform Volume Management of the floppy's presence.

OPTIONS

−d	Display the name of the default device to be ejected.
−f	Force the device to eject even if it is busy.
−n	Display the nickname to device name translation table.
−q	Query to see if the media is present.
device	Specify which device to **eject**, by the name it appears in the directory **/dev**.
nickname	Specify which device to **eject**, by its nickname as known to this command.

EXIT CODES

eject returns the following exit codes:

0	If the operation was successful or, with the **−q** option, the media *is* in the drive.
1	If the operation was unsuccessful or, with the **−q** option, the media is *not* in the drive.
2	If invalid flags were specified.
3	If an **ioctl( )** request failed.

4	Manually ejectable media is now okay to remove.

FILES

/dev/rfd0c	
/dev/rsr0	raw files
/dev/sr0	block files
/usr/lib/vold/eject_popup	popup used for manually ejected media

SEE ALSO

volcancel(1), **volcheck**(1), **volmissing**(1) **mount**(1M), **rmmount**(1M), **vold**(1M), **rmmount.conf**(4), **vold.conf**(4), **volfs**(7)

DIAGNOSTICS

A short help message is printed if an unknown flag is specified. A diagnostic is printed if the device name cannot be opened or does not support **eject**.

Device Busy	An attempt was made to eject a device that has a mounted filesystem. A warning message is printed when doing a forced eject of a mounted device.

BUGS

There should be a way to change the default on a per-user basis.

NAME

enable, disable – enable / disable LP printers

SYNOPSIS

/usr/bin/enable *printer ...*
/usr/bin/disable [**–c** **|** **–W**] [**–r** [*reason*]] *printer ...*

AVAILABILITY

SUNWlpu

DESCRIPTION

The **enable** command activates the named *printer*s, enabling them to print requests submitted by the **lp** command. If the printer is remote, the command will only enable the transfer of requests to the remote system; the **enable** command must be run again, on the remote system, to activate the printer. (Run **lpstat –p** to get the status of *printer*s.)

The **disable** command deactivates the named *printer*, disabling it from printing requests submitted by **lp**. By default, any requests that are currently printing on the designated printer(s) will be reprinted in their entirety either on the same printer or on another member of the same class of printers. If the printer is remote, this command will only stop the transmission of jobs to the remote system. The **disable** command must be run on the remote system to disable the printer. (Run **lpstat –p** to get the status of *printer*s.)

OPTIONS

Options for use with **disable** are:

–c	Cancel any requests that are currently printing on the designated printer(s). This option cannot be used with the **–W** option. If the printer is remote, the **–c** option will be silently ignored.
–W	Wait until the request currently being printed is finished before disabling the specified printer. This option cannot be used with the **–c** option. If the printer is remote, the **–W** option will be silently ignored.
–r *reason*	Assign a *reason* for the disabling of the printer(s). This *reason* applies to all *printer*s specified. This *reason* is reported by **lpstat –p**. *reason* must be enclosed in quotes if it contains blanks. The default reason is **unknown reason** for the existing printer, and **new printer** for a printer just added to the system but not yet enabled.

FILES

/var/spool/lp/∗

SEE ALSO

lp(1), **lpstat**(1)

NAME

env – obtain or alter environment variables for command execution

SYNOPSIS

env [–] [*name*=*value*] . . . [*command args*]

AVAILABILITY

SUNWcsu

DESCRIPTION

env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name*=*value* are merged into the inherited environment before the command is executed. The – flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

OPTIONS

–	Ignore the environment that would otherwise be inherited from the current shell. Restricts the environment for *command* to that specified by the arguments.
name=*value*	Set the environment variable *filename* to *value* and add it to the environment.

SEE ALSO

sh(1), **exec**(2), **profile**(4), **environ**(5)

NAME

eqn, neqn, checkeq – typeset mathematics test

SYNOPSIS

eqn [–**d***xy*] [–**f***n*] [–**p***n*] [–**s***n*] [*filename*] . . .
neqn [*filename*] . . .
checkeq [*filename*] . . .

AVAILABILITY

SUNWdoc

DESCRIPTION

eqn and **neqn** are language processors to assist in describing equations. **eqn** is a preprocessor for **troff**(1) and is intended for devices that can print **troff**'s output. **neqn** is a preprocessor for **nroff**(1) and is intended for use with terminals. Usage is almost always:

```
example% eqn filename ... | troff
example% neqn filename ... | nroff
```

If no *filename*s are specified, **eqn** and **neqn** read from the standard input. A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN**. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as ‘‘delimiters’’; subsequent text between delimiters is also treated as **eqn** input.

checkeq reports missing or unbalanced delimiters and **.EQ/.EN** pairs.

OPTIONS

–**d***xy*	Set equation delimiters set to characters *x* and *y* with the command-line argument. The more common way to do this is with **delim***xy* between **.EQ** and **.EN**. The left and right delimiters may be identical. Delimiters are turned off by **delim off** appearing in the text. All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched.
–**f***n*	Change font to *n* globally in the document. The font can also be changed globally in the body of the document by using the **gfont** directive.
–**p***n*	Reduce subscripts and superscripts by *n* point sizes from the previous size. In the absence of the –**p** option, subscripts and superscripts are reduced by 3 point sizes from the previous size.
–**s***n*	Change point size to *n* globally in the document. The point size can also be changed globally in the body of the document by using the **gsize** directive.

EQN LANGUAGE

Tokens within **eqn** are separated by braces, double quotes, tildes, circumflexes, SPACE, TAB, or NEWLINE characters. Braces { } are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (˜) represents a full SPACE in the output, circumflex (ˆ) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus ‘**x sub i**’ makes x_i , ‘**a sub i sup 2**’ produces a_i^2, and ‘**e sup {x sup 2 + y sup 2}**’ gives $e^{x^2+y^2}$.

Fractions are made with **over**: ‘**a over b**’ yields $\frac{a}{b}$.

sqrt makes square roots: ‘**1 over sqrt {ax sup 2 +bx+c}**’ results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim\limits_{n \to \infty} \sum\limits_0^n x_i$ is made with ‘**lim from {n→ inf } sum from 0 to n x sub i**’.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: ‘**left [x sup 2 + y sup 2 over alpha right] ˜=˜1**’ produces $\left[x^2 + \frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: ‘**pile {a above b above c}**’ produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: ‘**matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }**’ produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: ‘**x dot = f(t) bar**’ is $\dot{x} = \overline{f(t)}$, ‘**y dotdot bar ˜=˜ n under**’ is $\overline{\ddot{y}} = \underline{n}$, and ‘**x vec ˜=˜ y dyad**’ is $\vec{x} = \overleftrightarrow{y}$.

Sizes and font can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments −**s***n* and −**f***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define *thing* **%** *replacement* **%**

defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The **%** may be any character that does not occur in *replacement*.

Keywords like **sum** (∑), **int** (∫), **inf** (∞), and shorthands like **>=** (≥), → (→), and **!=** (≠) are recognized. Greek letters are spelled out in the desired case, as in **alpha** or **GAMMA**. Mathematical words like sin, cos, and log are made Roman automatically. **troff**(1) four-character escapes like \(bu (•) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with **troff** when all else fails.

SEE ALSO

tbl(1), **troff**(1), **ms**(5)

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in ‘**bold "12.3"**’.

NAME

error – insert compiler error messages at right source lines

SYNOPSIS

error [**–n**] [**–q**] [**–s**] [**–v**] [**–t** *suffixlist*] [**–I** *ignorefile*] [*filename*]

DESCRIPTION

error analyzes error messages produced by a number of compilers and language processors. It replaces the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously.

error looks at error messages, either from the specified file *filename* or from the standard input, and:

- Determines which language processor produced each error message.
- Determines the file name and line number of the erroneous line.
- Inserts the error message into the source file immediately preceding the erroneous line.

Error messages that can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. **error** touches source files only after all input has been read.

error is intended to be run with its standard input connected with a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into **error.** For example, when using the **csh** syntax, the following command analyzes all the error messages produced by whatever programs **make**(1S) runs when making lint:

```
example% make –s lint | & error
```

error knows about the error messages produced by: **as**(1), **cpp**(1), **ld**(1), **cc**(1B), **make**(1S) and other compilers. For all languages except Pascal, error messages are restricted to one line. Some error messages refer to more than one line in more than one file, in which case **error** duplicates the error message and inserts it in all the appropriate places.

OPTIONS

–n Do *not* touch any files; all error messages are sent to the standard output.

–q **error** asks whether the file should be touched. A '**y**' or '**n**' to the question is necessary to continue. Absence of the **–q** option implies that all referenced files (except those referring to discarded error messages) are to be touched.

–s Print out statistics regarding the error categorization.

–v After all files have been touched, overlay the visual editor **vi** with it set up to edit all files touched, and positioned in the first touched file at the first error. If **vi**(1) can't be found, try **ex**(1) or **ed**(1) from standard places.

–t *suffixlist*
Take the following argument as a suffix list. Files whose suffices do not appear in the suffix list are not touched. The suffix list is dot separated, and '∗' wildcards work. Thus the suffix list:

.c.y.f∗.h

allows **error** to touch files ending with '**.c**', '**.y**', '**.f∗**' and '**.h**'.

error catches interrupt and terminate signals, and terminates in an orderly fashion.

EXAMPLES

In the following C shell **(/usr/bin/csh)** example, **error** takes its input from the FORTRAN complier:

example% f77 –c *any***.f |& error"** *options*

Here is the same example using the Korn shell **(/usr/bin/ksh)**:

example% f77 –c *any***.f 2>&1 | error"** *options*

USAGE

error does one of six things with error messages.

synchronize	Some language processors produce short errors describing which file they are processing. **error** uses these to determine the file name for languages that do not include the file name in each error message. These synchronization messages are consumed entirely by **error**.
discard	Error messages from **lint** that refer to one of the two **lint** libraries, **/usr/lib/lint/llib-lc** and **/usr/lib/lint/llib-port** are discarded, to prevent accidentally touching these libraries. Again, these error messages are consumed entirely by **error**.
nullify	Error messages from **lint** can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named **.errorrc** in the user's home directory, or from the file named by the **–I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.
not file specific	Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They are not inserted into any source file.
file specific	Error messages that refer to a specific file but to no specific line are written to the standard output when that file is touched.
true errors	Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are inserted into source files. Other error messages are consumed entirely by **error** or are written to the standard output. **error** inserts the error messages into the source file on the line preceding the line number in the error message.

Each error message is turned into a one line comment for the language, and is internally flagged with the string **###** at the beginning of the error, and **%%%** at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

FILES

~/.errorrc	function names to ignore for **lint** error messages
/dev/tty	user's teletype

SEE ALSO

as(1), **cpp**(1), **csh**(1), **ed**(1), **ex**(1), **ld**(1), **vi**(1), **cc**(1B), **make**(1S)

BUGS

Opens the tty-device directly for user input.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's error message format may cause **error** to not understand the error message.

error, since it is purely mechanical, will not filter out subsequent errors caused by ''floodgating'' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected, error puts them before. The alignment of the **|** marking the point of error is also disturbed by **error.**

error was designed for work on CRT 's at reasonably high speed. It is less pleasant on slow speed terminals, and was not designed for use on hardcopy terminals.

NAME

ex – text editor

SYNOPSIS

ex [– | –**s**] [–**l**] [–**L**] [–**R**] [–**r** [*filename*]] [–**t** *tag*] [–**v**] [–**V**] [–**x**] [–**w***n*] [–**C**]
[+*command* | –**c** *command*] *filename*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

ex is the root of a family of editors: **ex** and **vi**. **ex** is a superset of **ed**, with the most notable extension being a display editing facility. Display based editing is the focus of **vi**.

If you have a CRT terminal, you may wish to use a display based editor; in this case see **vi**(1), which is a command which focuses on the display-editing portion of **ex**.

For ed Users

If you have used **ed** you will find that, in addition to having all of the **ed** commands available, **ex** has a number of additional features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with **vi**. Generally, the **ex** editor uses far more of the capabilities of terminals than **ed** does, and uses the terminal capability data base (see **terminfo**(4)) and the type of the terminal you are using from the environment variable TERM to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using the **vi** command.

ex contains a number of features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Typing **ˆD** (control-D) causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just typing return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. **ex** gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files, unless you edited them, so that you do not accidentally overwrite a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor **recover** command (or –**r** *filename* option) to retrieve your work. This will get you back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

The editor has a group of buffers whose names are the ASCII lower-case letters (**a-z**). You can place text in these named buffers where it is available to be inserted elsewhere in the file. The contents of these buffers remain available when you begin editing a new file using the **edit** (**e**) command.

There is a command **&** in **ex** which repeats the last **substitute** command. In addition, there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore the case of letters in searches and substitutions. **ex** also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word ‘‘edit’’ if your document also contains the word ‘‘editor.’’

ex has a set of options which you can set to tailor it to your liking. One option which is very useful is the **autoindent** option that allows the editor to supply leading white space to align text automatically. You can then use **^D** as a backtab and space or tab to move forward to align new code easily.

Miscellaneous useful features include an intelligent **join** (**j**) command that supplies white space between joined lines automatically, commands **<** and **>** which shift groups of lines, and the ability to filter portions of the buffer through commands such as **sort**.

OPTIONS

– | –**s**	Suppress all interactive user feedback. This is useful when processing editor scripts.
–**l**	Set up for editing LISP programs.
–**L**	List the name of all files saved as the result of an editor or system crash.
–**R**	**Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.
–**r** *filename*	Edit *filename* after an editor or system crash. (Recovers the version of *filename* that was in the buffer when the crash occurred.)
–**t** *tag*	Edit the file containing the *tag* and position the editor at its definition.
–**v**	Start up in display editing state using **ex**. You can achieve the same effect by simply typing the –**ex** command itself.
–**V**	Verbose. Any non-tty input will be echoed on standard error. This may be useful when processing editor commands within shell scripts.
–**x**	Encryption option. Simulates the **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of the **crypt** command. The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the –**x** option.
–**w***n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
–**C**	Encryption option. Same as the –**x** option, except simulates the **C** command. The **C** command is like the **X** command, except that all text read

in is assumed to have been encrypted.

+*command* | **−c** *command*
Begin editing by executing the specified editor *command* (usually a search or positioning command).

ex States

Command	Normal and initial state. Input prompted for by ‘‘**:**’’. Your line kill character cancels a partial command.
Insert	Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert state normally is terminated by a line having only "**.**" on it, or, abnormally, with an interrupt.
Visual	Entered by typing **vi**; terminated by typing **Q** or ˆ\ (control-\).

ex Command Names and Abbreviations

abbrev	**ab**	map		set	**se**
append	**a**	mark	**ma**	shell	**sh**
args	**ar**	move	**m**	source	**so**
change	**c**	next	**n**	substitute	**s**
copy	**co**	number	**nu**	unabbrev	**unab**
delete	**d**	preserve	**pre**	undo	**u**
edit	**e**	print	**p**	unmap	**unm**
file	**f**	put	**pu**	version	**ve**
global	**g**	quit	**q**	visual	**vi**
insert	**i**	read	**r**	write	**w**
join	**j**	recover	**rec**	xit	**x**
list	**l**	rewind	**rew**	yank	**ya**

ex Commands

forced encryption	**C**	heuristic encryption	**X**
resubst	**&**	print next	CR
rshift	>	lshift	<
scroll	**ˆD**	window	**z**
shell escape	**!**		

ex Command Addresses

n	line *n*	/*pat*	next with *pat*
.	current	**?***pat*	previous with *pat*
$	last	*x*-*n*	*n* before *x*
+	next	*x*,*y*	*x* through *y*
–	previous	´*x*	marked with *x*
+*n*	*n* forward	´´	previous context
%	1,$		

Initializing options

EXINIT	place **set**’s here in environment variable
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option *x*
set no*x*	disable option *x*
set *x***=val**	give value *val* to option *x*
set	show changed options
set all	show all options
set *x***?**	show value of option *x*

Most useful options and their abbreviations

autoindent	**ai**	supply indent
autowrite	**aw**	write before changing files
directory		pathname of directory for temporary work files
exrc	**ex**	allow **vi** / **ex** to read the **.exrc** in the current directory. This option is set in the **EXINIT** shell variable or in the **.exrc** file in the **$HOME** directory.
ignorecase	**ic**	ignore case of letters in scanning
list		print **^I** for tab, $ at end
magic		treat **. [** ∗ special in patterns
modelines		first five lines and last five lines executed as **vi** / **ex** commands if they are of the form **ex:***command***:** or **vi:***command***:**
number	**nu**	number lines
paragraphs	**para**	macro names that start paragraphs
redraw		simulate smart terminal
report		informs you if the number of lines modified by the last command is greater than the value of the **report** variable
scroll		command mode lines
sections	**sect**	macro names that start sections
shiftwidth	**sw**	for **< >**, and input **^D**
showmatch	**sm**	to **)** and **}** as typed
showmode	**smd**	show insert mode in **vi**
slowopen	**slow**	stop updates during insert
term		specifies to **vi** the type of terminal being used (the default is the value of the environment variable **TERM**)
window		visual mode lines
wrapmargin	**wm**	automatic line splitting
wrapscan	**ws**	search around end (or beginning) of buffer

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[str]	any character in *str*
[^str]	any character not in *str*
[*x–y***]**	any character between *x* and *y*
*	any number of preceding characters

AUTHOR

vi and **ex** are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/lib/expreserve	preserve command
/usr/lib/exrecover	recover command
/usr/lib/exstrings	error messages
/usr/share/lib/terminfo/*	describes capabilities of terminals
/var/preserve/login	preservation directory (where **login** is the user's login)
$HOME/.exrc	editor startup file
./.exrc	editor startup file

SEE ALSO

ed(1), **edit**(1), **grep**(1), **sed**(1), **sort**(1), **vi**(1), **curses**(3X), **term**(4), **terminfo**(4)

Solaris Advanced User's Guide

NOTES

Several options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard (see **intro**(1)). The – option has been replaced by –**s**, a –**r** option that is not followed with an option-argument has been replaced by –**L**, and +*command* has been replaced by –**c** *command*.

The message **file too large to recover with –r option ,** which is seen when a file is loaded, indicates that the file can be edited and saved successfully, but if the editing session is lost, recovery of the file with the –**r** option will not be possible.

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

The **z** command prints the number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input / output errors do not print a name if the command line –**s** option is used.

The editing environment defaults to certain configuration options. When an editing session is initiated, **ex** attempts to read the **EXINIT** environment variable. If it exists, the editor uses the values defined in **EXINIT**, otherwise the values set in **$HOME/.exrc** are used. If **$HOME/.exrc** does not exist, the default values are used.

To use a copy of **.exrc** located in the current directory other than **$HOME**, set the *exrc* option in **EXINIT** or **$HOME/.exrc**. Options set in **EXINIT** can be turned off in a local **.exrc** only if *exrc* is set in **EXINIT** or **$HOME/.exrc**.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

exec, eval, source – shell built-in functions to execute other commands

SYNOPSIS

sh

exec [*argument*. . .]

eval [*argument*. . .]

csh

exec *command*

eval *argument* . . .

source [–**h**] *name*

ksh

† **exec** [*arg* . . .]

† **eval** [*arg* . . .]

DESCRIPTION

sh

The **exec** command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

The *argument*s to the **eval** built-in are read as input to the shell and the resulting command(s) executed.

csh

exec executes *command* in place of the current shell, which terminates.

eval reads its *argument*s as input to the shell and executes the resulting command(s). This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions.

source reads commands from *name*. **source** commands may be nested, but if they are nested too deeply the shell may run out of file descriptors. An error in a sourced file at any level terminates all nested **source** commands.

–**h**	Place commands from the file *name* on the history list without executing them.

ksh

With the **exec** built-in, if *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

The arguments to **eval** are read as input to the shell and the resulting command(s) executed.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.

3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the = sign and word splitting and file name generation are not performed.

SEE ALSO

csh(1), **ksh**(1), **sh**(1)

NAME

exit, return, goto – shell built-in functions to enable the execution of the shell to advance beyond its sequence of steps

SYNOPSIS

sh

exit [*n*]

csh

exit [**(***expr***)**]

return [*n*]

goto *label*

ksh

† **exit** [*n*]

† **return** [*n*]

DESCRIPTION

sh

exit will cause a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an EOF will also cause the shell to exit.)

return causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

csh

exit will cause the shell to exit, either with the value of the status variable or with the value specified by the expression *expr* .

The **goto** built-in uses a specified *label* as a search string amongst commands. The shell rewinds its input as much as possible and searches for a line of the form *label***:** possibly preceded by space or tab characters. Execution continues after the indicated line. It is an error to jump to a label that occurs between a **while** or **for** built-in command and its corresponding **end**.

ksh

exit will cause the shell to exit with the exit status specified by *n*. The value will be the least significant 8 bits of the specified status. If *n* is omitted then the exit status is that of the last command executed.When **exit** occurs when executing a trap, the last command refers to the command that executed before the trap was invoked. An end-of-file will also cause the shell to exit except for a shell which has the **ignoreeof** option (See **set** below) turned on.

return causes a shell function or **'.'** script to return to the invoking script with the return status specified by *n*. The value will be the least significant 8 bits of the specified status. If *n* is omitted then the return status is that of the last command executed. If **return** is invoked while not in a function or a **'.'** script, then it is the same as an **exit**.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.

4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

break(1), **csh**(1), **ksh**(1), **sh**(1)

NAME

expand, unexpand – expand TAB characters to SPACE characters, and vice versa

SYNOPSIS

expand [–*tabstop*] [–*tab1, tab2,. . ., tabn*] [*filename*. . .]

unexpand [–**a**] [*filename*. . .]

AVAILABILITY

SUNWesu

DESCRIPTION

expand copies *filename*s (or the standard input) to the standard output, with TAB characters expanded to SPACE characters. BACKSPACE characters are preserved into the output and decrement the column count for TAB calculations. **expand** is useful for preprocessing character files (before sorting, looking at specific columns, and so forth) that contain TAB characters.

unexpand copies *filename*s (or the standard input) to the standard output, putting TAB characters back into the data. By default, only leading SPACE and TAB characters are converted to strings of tabs, but this can be overridden by the –**a** option (see the **OPTIONS** section below).

OPTIONS

expand options are:

–*tabstop*	Specify as a single argument, sets TAB characters *tabstop* SPACE characters apart instead of the default 8.
–*tab1, tab2,. . ., tabn*	Set TAB characters at the columns specified by *tab1*. . .

unexpand options are:

–**a**	Insert TAB characters when replacing a run of two or more SPACE characters would produce a smaller output file.

ENVIRONMENT

LC_CTYPE determines how **expand** handles characters. When **LC_CTYPE** is set to a valid value, **expand** can display and handle text and filenames containing valid characters for that locale. **expand** can display and handle Extended Unix Code (EUC) characters where any character can be 1, 2, or 3 bytes wide. **expand** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

NAME

exportfs – translates exportfs options to share / unshare commands

SYNOPSIS

/usr/sbin/exportfs [**–aiuv**] [**–o** *options*] [*pathname*]

AVAILABILITY

SUNWcsu

DESCRIPTION

exportfs translates SunOS 4.x **exportfs** options to the corresponding **share** / **unshare** options and invokes **share** / **unshare** with the translated options.

With no options or arguments, **exportfs** invokes **share** to print out the list of all currently shared NFS filesystems.

exportfs is the BSD / Compatibility Package command of **share**(1M) and **unshare**(1M). Use **share**(1M) / **unshare**(1M) whenever possible.

OPTIONS

–a	Invokes **shareall**(1M), or if **–u** is specified, invokes **unshareall**(1M).
–i	Ignore options in **/etc/dfs/dfstab**.
–u	Invokes **unshare**(1M) on *pathname*.
–v	Verbose.
–o *options*	Specify a comma-separated list of optional characteristics for the filesystems being exported. **exportfs** translates *options* to **share**-equivalent options. (see **share**(1M) for information about individual options).

SEE ALSO

share(1M), **shareall**(1M), **unshare**(1M), **unshareall**(1M)

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr *arguments*

AVAILABILITY

SUNWcsu

DESCRIPTION

The *arguments* are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers. The length of the expression is limited to 512 characters.

USAGE

The operators and keywords are listed below. Characters that need to be escaped in the shell (see **sh**(1)) are preceded by \\. The list is in order of increasing precedence, with equal precedence operators grouped within **{}** symbols.

expr **\\|** *expr*
: returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr **\\&** *expr*
: returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { **=**, **\\>**, **\\>=**, **\\<**, **\\<=**, **!=** } *expr*
: returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { **+**, **–** } *expr*
: addition or subtraction of integer-valued arguments.

expr { **\\***, **/**, **%** } *expr*
: multiplication, division, or remainder of the integer-valued arguments.

expr **:** *expr*
: The matching operator **:** compares the first argument with the second argument, which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are ‘‘anchored’’ (that is, begin with **^**) and, therefore, **^** is not a special character, in that context. Normally, the matching operator returns the number of bytes matched (**0** on failure). Alternatively, the **\\(...\\)** pattern symbols can be used to return a portion of the first argument.

EXAMPLES

Add 1 to the shell variable **a**:

```
a=`expr $a + 1`
```

The following example emulates **basename**(1) — it returns the last segment of the path name **$a**. For **$a** equal to either **/usr/abc/file** or just **file**, the example returns **file**. (Watch out for **/** alone as an argument: **expr** takes it as the division operator; see the NOTES below.)

```
expr $a : ´.*/\(.*\)´ \| $a
```

Here is a better version of the previous example. The addition of the **//** characters eliminates any ambiguity about the division operator and simplifies the whole expression.

```
expr //$a : ´.*/\(.*\)´
```

Return the number of characters in **$VAR**:

```
expr $VAR : ´.*´
```

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **expr** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **expr** behaves.

LC_CTYPE

Determines how **expr** handles characters. When **LC_CTYPE** is set to a valid value, **expr** can display and handle text and filenames containing valid characters for that locale. **expr** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **expr** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

ed(1), **sh**(1), **environ**(5)

DIAGNOSTICS

As a side effect of expression evaluation, **expr** returns the following exit values:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

syntax error	for operator / operand errors
non-numeric argument	if arithmetic is attempted on such a string

NOTES

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an **=**, the command:

```
expr $a = ´=´
```

looks like:

```
expr = = =
```

as the arguments are passed to **expr** (and they are all taken as the **=** operator). The following works:

```
expr X$a = X=
```

NAME

expr – evaluate arguments as a logical, arithmetic, or string expression

SYNOPSIS

/usr/ucb/expr *argument*. . .

AVAILABILITY

SUNWscpu

DESCRIPTION

expr evaluates expressions as specified by its arguments. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument, so terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note: **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, two's-complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by '\'. The list is in order of increasing precedence, with equal precedence operators grouped within **{}** symbols.

expr \| *expr*
: Return the first *expr* if it is neither NULL nor **0**, otherwise returns the second *expr*.

expr **\&** *expr*
: Return the first *expr* if neither *expr* is NULL or **0**, otherwise returns **0**.

expr { **=,** \>**,** \>= , **\<,** **\<=,** **!=** } *expr*
: Return the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { **+**, **–** } *expr*
: Addition or subtraction of integer-valued arguments.

expr { *, **/**, **%** } *expr*
: Multiplication, division, or remainder of the integer-valued arguments.

string **:** *regular-expression*
match *string regular-expression*
: The two forms of the matching operator above are synonymous. The matching operators **:** and **match** compare the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are "anchored" (treated as if they begin with **^**) and, therefore, **^** is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the **\(. . . \)** pattern symbols can be used to return a portion of the first argument.

substr *string integer-1 integer-2*

Extract the substring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1* has a value greater than the length of *string*, **expr** returns a null string. If you try to extract more characters than there are in *string*, **expr** returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as **expr** tends to run forever in these cases.

index *string character-list*

Report the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

length *string*

Return the length (that is, the number of characters) of *string*.

(expr) Parentheses may be used for grouping.

EXAMPLES

1.
```
a=‘expr $a + 1‘
```
Adds 1 to the shell variable **a**.

2.
```
# ’For $a equal to either "/usr/abc/file" or just "file"’
expr $a : ’.*/\(.*\)’ \| $a
```
Returns the last segment of a path name (that is, the filename part). Watch out for **/** alone as an argument: *expr* will take it as the division operator (see BUGS below).

3.
```
# A better representation of example 2.
expr //$a : ’.*/\(.*\)’
```
The addition of the **//** characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4.
```
expr $VAR : ’.*’
```
Returns the number of characters in **$VAR**.

SEE ALSO

ed(1), **sh**(1), **test**(1)

EXIT CODES

expr returns the following exit codes:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

DIAGNOSTICS

syntax error	for operator / operand errors
non-numeric argument	if arithmetic is attempted on such a string
division by zero	if an attempt to divide by zero is made

BUGS

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an **=**, the command:

```
expr  $a  =  '='
```

looks like:

```
expr  =  =  =
```

as the arguments are passed to **expr** (and they will all be taken as the **=** operator). The following works:

```
expr  X$a  =  X=
```

Note: the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

```
example% expr index expurgatorious length
syntax error
example%
```

generates the '**syntax error**' message as shown instead of the value **1** as you might expect.

NAME

exstr – extract strings from source files

SYNOPSIS

exstr *filename . . .*
exstr –e *filename . . .*
exstr –r [**–d**] *filename . . .*

DESCRIPTION

The **exstr** utility is used to extract strings from C-language source files and replace them by calls to the message retrieval function (see **gettxt**(3C)). This utility will extract all character strings surrounded by double quotes, not just strings used as arguments to the **printf** command or the **printf** routine. In the first form, **exstr** finds all strings in the source files and writes them on the standard output. Each string is preceded by the source file name and a colon.

The first step is to use **exstr –e** to extract a list of strings and save it in a file. Next, examine this list and determine which strings can be translated and subsequently retrieved by the message retrieval function. Then, modify this file by deleting lines that can't be translated and, for lines that can be translated, by adding the message file names and the message numbers as the fourth (*msgfile*) and fifth (*msgnum*) entries on a line. The message files named must have been created by **mkmsgs**(1) and exist in **/usr/lib/locale/***locale***/LC_MESSAGES**. (The directory *locale* corresponds to the language in which the text strings are written; see **setlocale**(3C)). The message numbers used must correspond to the sequence numbers of strings in the message files.

Now use this modified file as input to **exstr –r** to produce a new version of the original C-language source file in which the strings have been replaced by calls to the message retrieval function **gettxt**( ). The *msgfile* and *msgnum* fields are used to construct the first argument to **gettxt**( ). The second argument to **gettxt**( ) is printed if the message retrieval fails at run-time. This argument is the null string, unless the **–d** option is used.

This utility cannot replace strings in all instances. For example, a static initialized character string cannot be replaced by a function call. A second example is that a string could be in a form of an escape sequence which could not be translated. In order not to break existing code, the files created by invoking **exstr –e** must be examined and lines containing strings not replaceable by function calls must be deleted. In some cases the code may require modifications so that strings can be extracted and replaced by calls to the message retrieval function.

OPTIONS

–e Extract a list of strings from the named C-language source files, with positional information. This list is produced on standard output in the following format:

file:line:position:msgfile:msgnum:string

file	the name of a C-language source file
line	line number in the file
position	character position in the line
msgfile	null
msgnum	null
string	the extracted text string

Normally you would redirect this output into a file. Then you would edit this file to add the values you want to use for *msgfile* and *msgnum*:

msgfile	the file that contains the text strings that will replace *string*. A file with this name must be created and installed in the appropriate place by the **mkmsgs**(1) utility.
msgnum	the sequence number of the string in *msgfile*.

The next step is to use **exstr –r** to replace *string*s in *file*.

–r Replace strings in a C-language source file with function calls to the message retrieval function **gettxt**().

–d This option is used together with the **–r** option. If the message retrieval fails when **gettxt**() is invoked at run-time, then the extracted string is printed. You would use the capability provided by **exstr** on an application program that needs to run in an international environment and have messages print in more than one language. **exstr** replaces text strings with function calls that point at strings in a message data base. The data base used depends on the run-time value of the **LC_MESSAGES** environment variable (see **environ**(5)).

EXAMPLES

The following examples show uses of **exstr**.

Assume that the file **example.c** contains two strings:

```
main()
{
        printf("This is an example\n");
        printf("Hello world!\n");
}
```

The **exstr** utility, invoked with the argument **example.c** extracts strings from the named file and prints them on the standard output.

```
example% exstr example.c
```

produces the following output:

```
example.c:This is an example\n
example.c:Hello world!\n
```

```
example% exstr –e example.c > example.stringsout
```

produces the following output in the file **example.stringsout**:

```
example.c:3:8:::This is an example\n
example.c:4:8:::Hello world!\n
```

You must edit **example.stringsout** to add the values you want to use for the *msgfile* and *msgnum* fields before these strings can be replaced by calls to the retrieval function. If **UX** is the name of the message file, and the numbers **1** and **2** represent the sequence number of the strings in the file, here is what **example.stringsout** looks like after you add this information:

```
example.c:3:8:UX:1:This is an example\n
example.c:4:8:UX:2:Hello world!\n
```

The **exstr** utility can now be invoked with the **–r** option to replace the strings in the source file by calls to the message retrieval function **gettxt**().

```
example% exstr –r example.c <example.stringsout >intlexample.c
```

produces the following output:

```
extern char *gettxt();
main()
{
        printf(gettxt("UX:1", ""));
        printf(gettxt("UX:2", ""));
}
```

```
example% exstr –rd example.c <example.stringsout >intlexample.c
```

uses the extracted strings as a second argument to **gettxt**().

```
extern char *gettxt();
main()
{
        printf(gettxt("UX:1", "This is an example\n"));
        printf(gettxt("UX:2", "Hello world!\n"));
}
```

FILES

/usr/lib/locale/*locale***/LC_MESSAGES/*** files created by **mkmsgs**(1)

SEE ALSO

gettxt(1), **mkmsgs**(1), **printf**(1), **srchtxt**(1), **gettxt**(3C), **printf**(3S), **setlocale**(3C), **environ**(5)

DIAGNOSTICS

The error messages produced by **exstr** are intended to be self-explanatory. They indicate errors in the command line or format errors encountered within the input file.

NAME

face – executable for the Framed Access Command Environment Interface

SYNOPSIS

face [**–i** *init_file*] [**–c** *command_file*] [**–a** *alias_file*] [*filename*. . | . |]

DESCRIPTION

filename is the full pathname of the file describing the object to be opened initially, and must follow the naming convention **Menu.***xxx* for a menu, **Form.***xxx* for a form, and **Text.***xxx* for a text file, where *xxx* is any string that conforms to the UNIX system file naming conventions. The Form and Menu Language Interpreter (FMLI) descriptor **lifetime** will be ignored for all frames opened by argument to **face**. These frames have a lifetime of **immortal** by default. If *filename* is not specified on the command line, the AT&T FACE Menu will be opened along with those objects specified by the **LOGINWIN** environment variables. These variables are found in the user's **.environ** file.

OPTIONS

–**a** *alias_file*	Alias file.
–**c** *command_file*	Command file.
–**i** *init_file*	Initial file.

FILES

$HOME/pref/.environ

SEE ALSO

env(1)

DIAGNOSTICS

The **face** command will exit with a non-zero exit code if the user is not properly set up as a FACE user.

NAME

factor – obtain the prime factors of a number

SYNOPSIS

factor [*integer*]

AVAILABILITY

SUNWesu

DESCRIPTION

When you use **factor** without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. **factor** exits if it encounters a zero or any non-numeric character.

If you invoke **factor** with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. **factor** will take this time when n is prime or the square of a prime.

DIAGNOSTICS

factor prints the error message, **Ouch**, for input out of range or for garbage input.

NAME

fastboot, fasthalt – reboot / halt the system without checking the disks

SYNOPSIS

/usr/ucb/fastboot [*boot-options*]

/usr/ucb/fasthalt [*halt-options*]

AVAILABILITY

SUNWscpu

DESCRIPTION

fastboot and **fasthalt** are shell scripts that invoke **reboot** and **halt** with the proper arguments.

These commands are provided for compatibility only.

SEE ALSO

fsck(1M), **halt**(1M), **init**(1M), **reboot**(1M), **init.d**(4)

NAME

fdformat – format floppy diskette

SYNOPSIS

fdformat [**–dDeEfHlLmMUqvx**] [**–b** *label*] [**–B** *filename*] [**–t** *dostype*] [*devname*]

AVAILABILITY

SUNWcsu

DESCRIPTION

fdformat is a utility for formatting diskettes. All new, blank diskettes must be formatted before they can be used. **fdformat** formats and verifies each track on the diskette, and indicates whether any bad sectors were encountered. All existing data on the diskette, if any, is destroyed by formatting.

By default, **fdformat** uses the configured capacity of the drive to format the diskette. A 3.5" high-density drive uses diskettes with a formatted capacity of 1.44 megabytes. A 5.25" high-density drive uses diskettes with a formatted capacity of 1.2 megabytes. In either case, a density option does not have to be specified to **fdformat**. However, a density option must be specified when using a diskette with a lower capacity than the drive's default. Use the –**H** option to format high-density diskettes (1.44-megabyte capacity) in an extra-high-density (ED) drive. Use the –**D** option, the –**l** option, or the –**L** option to format double-density (or "low-density") diskettes (720KB capacity) in an HD or ED drive. To format medium-density diskettes (1.2-megabyte capacity), use the –**M** option with –**t nec** (this is the same as using the –**m** option with –**t nec**).

Extended density uses double-sided, extended-density (or extra-high-density) (DS / ED) diskettes. Medium and high densities use the same media: double-sided, high-density (DS / HD) diskettes. Double ("low") density uses double-sided, double-density (DS / DD) diskettes. Substituting diskettes of one density for diskettes of either a higher or lower density generally will not work. Data integrity cannot be assured whenever a diskette is formatted to a capacity not matching its density.

fdformat writes new identification and data fields for each sector on all tracks unless the –**x** option is specified. If the –**v** option is specified, each sector on all tracks is written and read back. Otherwise, a read of each sector is performed immediately after the track is formatted.

After formatting and verifying, **fdformat** writes an operating-system label on block 0. Use the –**t dos** option (same as the –**d** option) to put an MS-DOS file system on the diskette after format is done or use the –**t nec** option with the –**M** option (same as the –**m** option) to put an NEC-DOS file system on the diskette. Otherwise, **fdformat** writes a SunOS label on the diskette in block 0. The label is required on Solaris systems if you intend to put a UNIX file system on the diskette.

OPTIONS

–D	Format a 720KB (3.5") or 360KB (5.25") double-density diskette (same as the –**l** or –**L** options). This is the default for double-density type drives. It is needed if the drive is a high- or extended-density type.
–e	Eject the diskette when done. (This feature is not available on all systems).
–E	Format a 2.88-megabyte (3.5") extended-density diskette. This is the default for extended-density type drives.

–**f**	Force. Do not ask for confirmation before starting format.
–**H**	Format a 1.44-megabyte (3.5") or 1.2-megabyte (5.25") high-density diskette. This is the default for high-density type drives; it is needed if the drive is the extended-density type.
–**M**	Write a 1.2-megabyte (3.5") medium-density format on a high-density diskette (use only with the –**t nec** option). This is the same as using –**m**. (This feature is not available on all systems.)
–**U**	**umount** any file systems on the floppy and then format.
–**q**	Quiet; do not print status messages.
–**v**	Verify each block of the diskette after the format.
–**x**	Skip the format, and only write a SunOS label or an MS-DOS file system.
–**b** *label*	Label the diskette with volume *label*. A SunOS volume label is restricted to 8 characters. A DOS volume label is restricted to 11 upper-case characters.
–**B** *filename*	Install special boot loader in *filename* on an MS-DOS diskette. This option is only meaningful when the –**d option** (or **-t dos**) is also specified.
–**t dos**	Install an MS-DOS file system and boot sector on the disk after formatting. This is equivalent to the DOS format command or the –**d** option.
–**t nec**	Install an NEC-DOS file system and boot sector on the disk after formatting. This should be used only with the –**M** option. (This feature is not available on all systems).
devname	Replace *devname* with **rdiskette0** (systems without Volume Management) or **floppy0** (systems with Volume Management) to use the first drive or **rdiskette1** (systems without Volume Management) or **floppy1** (systems with Volume Management) to use the second drive. If *devname* is omitted, the default drive, if one exists, will be used.

The following options are provided for compatibility with previous versions of **fdformat**; their use is discouraged.

–**d**	Format an MS-DOS floppy diskette (same as –**t dos**). This is equivalent to the MS-DOS **FORMAT** command.
–**l**	Format a 720KB (3.5") or 360KB (5.25") double-density diskette (same as –**D** or –**L**). This is the default for double-density type drives; it is needed if the drive is the high- or extended-density type.
–**L**	Format a 720KB (3.5") or 360KB (5.25") double-density diskette (same as –**l** or –**D**). This is the default for double-density type drives; it is needed if the drive is the high- or extended-density type.
–**m**	Write a 1.2-megabyte (3.5") medium-density format on a high-density diskette (use only with the –**t nec** option). This is the same as using –**M**. (This feature is not available on all systems.)

FILES

/vol/dev/diskette0	Directory providing block device access for the media in floppy drive 0.
/vol/dev/rdiskette0	Directory providing character device access for the media in floppy drive 0.
/vol/dev/aliases/floppy0	Symbolic link to the character device for the media in floppy drive 0.
/dev/rdiskette	Directory providing character device access for the media in the primary floppy drive, usually drive 0.

SEE ALSO

cpio(1), **eject**(1), **tar**(1), **volcancel**(1), **volcheck**(1), **volmissing**(1), **mount**(1M), **newfs**(1M), **rmmount**(1M), **vold**(1M), **rmmount.conf**(4), **vold.conf**(4), **pcfs**(7), **volfs**(7)

x86 Only

fd(7)

NOTES

A diskette containing a ufs file system created on a SPARC (big endian) system (by using **fdformat** and **newfs**(1M)) is not identical to a diskette containing a ufs file system created on an x86 (little endian) system. Do not interchange ufs diskettes between these platforms; use **cpio**(1) or **tar**(1) to transfer files on diskettes between them.

A diskette formatted using the –**t dos** option (or –**d**) for MS-DOS will not have the necessary system files, and is therefore not bootable. Trying to boot from it on a PC will result in the following message:

```
Non-System disk or disk error
Replace and strike any key when ready
```

BUGS

Currently, bad sector mapping is not supported on floppy diskettes. Therefore, a diskette is unusable if **fdformat** finds an error (bad sector).

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [**–bchilnsvx**] [**–e** *special string*] [**–f** *filename*] [*strings*] [*filename* ...]

AVAILABILITY

SUNWcsu

DESCRIPTION

fgrep (fast **grep**) searches files for a character string and prints all lines that contain that string. **fgrep** is different from **grep**(1) and **egrep**(1) because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters **$**, **∗**, **[**, **ˆ**, **|** , **(**, **)**, and \ are interpreted literally by **fgrep**, that is, **fgrep** does not recognize full regular expressions as does **egrep**. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes ′...′.

If no files are specified, **fgrep** assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

OPTIONS

–b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
–c	Print only a count of the lines that contain the pattern.
–h	Suppress printing of filenames when searching multiple files.
–i	Ignore upper/lower case distinction during comparisons.
–l	Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
–n	Precede each line by its line number in the file (first line is 1).
–s	Work silently, that is, display nothing except error messages. This is useful for checking the error status.
–v	Print all lines except those that contain the pattern.
–x	Print only lines matched entirely.
–e *special_string*	Search for a *special string* (*string* begins with a –).
–f *filename*	Take the list of *strings* from *filename*.

ENVIRONMENT

If any of the **LC_∗** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **fgrep** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_∗** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **fgrep** behaves.

LC_CTYPE

Determines how **fgrep** handles characters. When **LC_CTYPE** is set to a valid value, **fgrep** can display and handle text and filenames containing valid characters for that locale. **fgrep** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **fgrep** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. / English).

SEE ALSO

ed(1), **egrep**(1), **grep**(1), **sed**(1), **sh**(1), **environ**(5)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

NOTES

Ideally there should be only one **grep** command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **<stdio.h>**.

NAME

file – determine file type

SYNOPSIS

file [–**h**] [–**m** *mfilename*] [–**f** *ffilename*] *argument* ...
file [–**h**] [–**m** *mfilename*] –**f** *ffilename*
file –**c** [–**m** *mfilename*]

AVAILABILITY

SUNWcsu

DESCRIPTION

file performs a series of tests on each file supplied by *argument* and, optionally, on each file supplied in *ffilename* in an attempt to classify it. If *argument* appears to be a text file, **file** examines the first 512 bytes and tries to guess its programming language. If *argument* is an executable **a.out**, **file** prints the version stamp, provided it is greater than 0. If *argument* is a symbolic link, by default the link is followed and **file** tests the file that the symbolic link references.

file uses **/etc/magic** to identify files that have a magic number. A magic number is a numeric or string constant that indicates the file type. Commentary at the beginning of **/etc/magic** explains its format.

OPTIONS

–**c**	Check the magic file for format errors. For reasons of efficiency, this validation is normally not carried out.
–**h**	Do not follow symbolic links.
–**f** *ffilename*	*ffilename* contains the names of the files to be examined.
–**m** *mfilename*	Use *mfilename* as an alternate magic file, instead of **/etc/magic .**

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **file** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **file** behaves.

LC_CTYPE
Determines how **file** handles characters. When **LC_CTYPE** is set to a valid value, **file** can display and handle text and filenames containing valid characters for that locale. **file** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **file** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/etc/magic

SEE ALSO

filehdr(4), **magic**(4), **environ**(5),

DIAGNOSTICS

If the **−h** option is specified and *argument* is a symbolic link, **file** prints the error message:

> **symbolic link to** *argument*

NAME

file – determine the type of a file by examining its contents

SYNOPSIS

/usr/ucb/file [**–f** *ffile*] [**–cL**] [**–m** *mfile*] *filename*. . .

AVAILABILITY

SUNWscpu

DESCRIPTION

file performs a series of tests on each *filename* in an attempt to determine what it contains. If the contents of a file appear to be ASCII text, **file** examines the first 512 bytes and tries to guess its language.

file uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type.

OPTIONS

–c	Check for format errors in the magic number file. For reasons of efficiency, this validation is not normally carried out. No file type-checking is done under **–c**.
–f *ffile*	Get a list of filenames to identify from *ffile.*
–L	If a file is a symbolic link, test the file the link references rather than the link itself.
–m *mfile*	Use *mfile* as the name of an alternate magic number file.

EXAMPLES

This example illustrates the use of **file** on all the files in a specific user's directory:

```
example% pwd
/usr/blort/misc
example% /usr/ucb/file  *
code:                  mc68020 demand paged executable
code.c:                c program text
counts:                ascii text
doc:                   roff, nroff , or eqn input text
empty.file:            empty
libz:                  archive random library
memos:                 directory
project:               symbolic link to /usr/project
script:                executable shell script
titles:                ascii text
s5.stuff:              cpio archive
example%
```

ENVIRONMENT

The environment variables **LC_CTYPE**, **LANG**, and **LC_default** control the character classification throughout **file**. On entry to **file**, these environment variables are checked in the following order: **LC_CTYPE**, **LANG**, and **LC_default.** When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**. When none of the values is valid, the shell character classification defaults to the POSIX.1 “C” locale.

FILES

/etc/magic

SEE ALSO

magic(4)

BUGS

file often makes mistakes. In particular, it often suggests that command files are C programs.

file does not recognize Pascal or LISP.

NAME

find – find files

SYNOPSIS

find *path-name-list expression*

AVAILABILITY

SUNWcsu

DESCRIPTION

find recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, one or more path names) seeking files that match a Boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where **+***n* means more than *n*, –*n* means less than *n* and *n* means exactly *n*.

USAGE

Expressions

Valid expressions are:

–**name** *pattern*	True if *pattern* matches the current file name. Normal shell file name generation characters (see **sh**(1)) may be used. A backslash (\ ) is used as an escape character within the pattern. The pattern should be escaped or quoted when **find** is invoked from the shell.
–**perm** [**-**]*onum*	True if the file permission flags exactly match the octal number *onum* (see **chmod**(1)). If *onum* is prefixed by a minus sign (–), only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.
–**size** *n*[**c**]	True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.
–**atime** *n*	True if the file was accessed *n* days ago. The access time of directories in *path-name-list* is changed by **find** itself.
–**ls**	Always true; prints current pathname together with its associated statistics. These include (respectively) inode number, size in kilobytes (1024 bytes), protection mode, number of hard links, user, group, size in bytes, and modification time. If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by ‘→’. The format is identical to that of **ls** –**gilds** (see **ls**(1)). Note: Formatting is done internally, without executing the **ls** program.
–**mtime** *n*	True if the file’s data was modified *n* days ago.
–**ctime** *n*	True if the file’s status was changed *n* days ago.
–**exec** *command*	True if the executed *command* returns a zero value as exit status. The end of *command* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current path name.
–**ok** *command*	Like –**exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.
-cpio *device*	Always true; write the current file on *device* in **cpio** format (5120-byte

	records).
-ncpio *device*	Always true; write the current file on *device* in **cpio -c** format (5120 byte records).
–**print**	Always true; causes the current path name to be printed.
–**newer** *filename*	True if the current file has been modified more recently than the argument *filename*.
–**depth**	Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when **find** is used with **cpio**(1) to transfer files that are contained in directories without write permission.
–**mount**	Always true; restricts the search to the file system containing the directory specified. Does not list mount points to other file systems.
-xdev	Same as the **-mount** option.
–**local**	True if the file system type is not a remote file system type as defined in the **/etc/dfs/fstypes** file. **nfs** is used as the default remote filesystem type if the **/etc/dfs/fstypes** file is not present.
(*expression* **)**	True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).
–**type** *c*	True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **l**, **p**, or **f** for block special file, character special file, directory, symbolic link, fifo (named pipe), or plain file, respectively.
–**follow**	Always true; causes symbolic links to be followed. When following symbolic links, **find** keeps track of the directories visited so that it can detect infinite loops; for example, such a loop would occur if a symbolic link pointed to an ancestor. This expression should not be used with the –**type l** expression.
–**links** *n*	True if the file has *n* links.
–**user** *uname*	True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.
–**nouser**	True if the file belongs to a user not in the **/etc/passwd** file.
–**group** *gname*	True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.
–**nogroup**	True if the file belongs to a group not in the **/etc/group** file.
–**fstype** *type*	True if the filesystem to which the file belongs is of type *type*.
–**inum** *n*	True if the file has inode number *n*.
–**prune**	Always yields true. Do not examine any directories or files in the directory structure below the *pattern* just matched. See the examples, below.

Primaries

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (**!** is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (–**o** is the *or* operator).

Note: When you use **find** in conjunction with **cpio**, if you use the –**L** option with **cpio** then you must use the –**follow** expression with **find** and vice versa. Otherwise there will be undesirable results.

EXAMPLES

Remove all files in your home directory named **a.out** or ***.o** that have not been accessed for a week:

```
example% find $HOME \( –name a.out –o –name ′*.o′ \) –atime +7 –exec rm {} \;
```

Recursively print all file names in the current directory and below, but skipping SCCS directories:

```
example% find . –name SCCS –prune –o –print
```

Recursively print all file names in the current directory and below, skipping the contents of SCCS directories, but printing out the SCCS directory name:

```
example% find . –print –name SCCS –prune
```

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **find** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **find** behaves.

LC_CTYPE

Determines how **find** handles characters. When **LC_CTYPE** is set to a valid value, **find** can display and handle text and filenames containing valid characters for that locale. **find** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **find** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/etc/passwd	password file
/etc/group	group file
/etc/dfs/fstypes	file that registers distributed file system packages

SEE ALSO

chmod(1), **ls**(1), **sh**(1), **test**(1), **stat**(2), **umask**(2), **environ**(5)

WARNINGS

The following options are obsolete and will not be supported in future releases:

-cpio *device*	Always true; write the current file on *device* in **cpio** format (5120-byte records).
-ncpio *device*	Always true; write the current file on *device* in **cpio -c** format (5120 byte records).

NOTES

When using **find** to determine files modified within a range of time, one must use the **?time** argument BEFORE the –**print** argument otherwise **find** will give all files.

NAME

finger – display information about local and remote users

SYNOPSIS

finger [–**bfhilmpqsw**] [*username*. . .]

finger [–**l**] [*username@hostname*1[*@hostname*2. . .*@hostname*n] . . .]

finger [–**l**] [*@hostname*1[*@hostname*2. . .*@hostname*n] . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

By default, the **finger** command displays in multi-column format the following information about each logged-in user:

- user name
- user's full name
- terminal name (prepended with a '∗' (asterisk) if write-permission is denied)
- idle time
- login time
- host name, if logged in remotely

Idle time is in minutes if it is a single integer, in hours and minutes if a ':' (colon) is present, or in days and hours if a 'd' is present.

When one or more *username* arguments are given, more detailed information is given for each *username* specified, whether they are logged in or not. *username* must be that of a local user, and may be a first or last name, or an account name. Information is presented in multi-line format as follows:

- the user name and the user's full name
- the user's home directory and login shell
- time the user logged in if currently logged in, or the time the user last logged in; and the terminal or host from which the user logged in
- last time the user received mail, and the last time the user read mail
- the first line of the **$HOME/.project** file, if it exists
- the contents of the **$HOME/.plan** file, if it exists

If the arguments *username@hostname*1[*@hostname*2...*@hostname*n] or *@hostname*1[*@hostname*2...*@hostname*n] are used, the request is sent first to *hostname*n and forwarded through each *hostname*n-1 to *hostname*1. The program uses the **finger user information protocol** (see RFC 1288) to query that remote host for information about the named user (if *username* is specified), or about each logged-in user. The information displayed is server dependent.

OPTIONS

The *username@hostname* form supports only the –**l** option.

–**b**	Suppress printing the user's home directory and shell in a long format printout.
–**f**	Suppress printing the header that is normally printed in a non-long format printout.
–**h**	Suppress printing of the **.project** file in a long format printout.
–**i**	Force "idle" output format, which is similar to short format except that only the

login name, terminal, login time, and idle time are printed.

−l	Force long output format.
−m	Match arguments only on user name (not first or last name).
−p	Suppress printing of the **.plan** file in a long format printout.
−q	Force quick output format, which is similar to short format except that only the login name, terminal, and login time are printed.
−s	Force short output format.
−w	Suppress printing the full name in a short format printout.

FILES

$HOME/. plan	user's plan
$HOME/. project	user's projects
/etc/passwd	password file
/var/adm/lastlog	time of last login
/var/adm/utmp	accounting

SEE ALSO

passwd(1), **who**(1), **whois**(1)

NOTES

The **finger user information protocol** limits the options that may be used with the remote form of this command.

NAME

fmlcut – cut out selected fields of each line of a file

SYNOPSIS

fmlcut –**c***list* [*filename* . . .]
fmlcut –**f** *list* [–**d** *char*] [–**s**] [*filename* . . .]

DESCRIPTION

The **fmlcut** function cuts out columns from a table or fields from each line in *filename*; in database parlance, it implements the projection of a relation. **fmlcut** can be used as a filter; if *filename* is not specified or is –, the standard input is read. *list* specifies the fields to be selected. Fields can be fixed length (character positions) or variable length (separated by a field delimiter character), depending on whether –**c** or –**f** is specified.

Note: Either the –**c** or the –**f** option must be specified.

OPTIONS

list	A comma-separated list of integer field numbers (in increasing order), with optional – to indicate ranges. For example: **1,4,7; 1**–**3,8;** –**5,10** (short for **1**–**5,10**); or **3**– (short for third through last field).
–**c***list*	If –**c** is specified, *list* specifies character positions (for instance, –**c1**–**72** would pass the first 72 characters of each line). Note: No space intervenes between –**c** and *list.*
–**f***list*	If –**f** is specified, *list* is a list of fields assumed to be separated in the file by the default delimiter character, **TAB,** or by *char* if the –**d** option is specified. For example, –**f1,7** copies the first and seventh field only. Lines with no delimiter characters are passed through intact (useful for table subheadings), unless –**s** is specified. Note: No space intervenes between –**f** and *list.* The following options can be used if you have specified –**f**.

–**d***char*	If –**d** is specified, *char* is the field delimiter. Space or other characters with special meaning to FMLI must be quoted. Note: No space intervenes between –**d** and *char* . The default field delimiter is **TAB.**
–**s**	Suppresses lines with no delimiter characters. If –**s** is not specified, lines with no delimiters will be passed through untouched.

EXAMPLES

The following example gets the login IDS and names.

```
example% fmlcut –d: –f1,5 /etc/passwd
```

The next example gets the current login name.

```
example% `who am i | fmlcut –f1 –d" "`
```

SEE ALSO

fmlgrep(1F)

DIAGNOSTICS

fmlcut returns the following exit values:

0	when the selected field is successfully cut out
2	on syntax errors

The following error messages may be displayed on the FMLI message line:

ERROR: line too long
A line has more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c / f option
Missing −**c** or −**f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

ERROR: no fields
The *list* is empty.

ERROR: no delimiter
Missing *char* on −**d** option.

NOTES

fmlcut cannot correctly process lines longer than 1023 characters, or lines with no new-line character.

NAME

fmlexpr – evaluate arguments as an expression

SYNOPSIS

fmlexpr *arguments*

DESCRIPTION

The **fmlexpr** function evaluates its arguments as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to FMLI must be escaped. Note: **30** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within **{}** symbols.

USAGE

Expressions

expr \\| *expr*	Returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.
expr \\**&** *expr*	Returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.
expr { **=**, \\**>**, \\**>=**, **<**, \**<=**, **!=** } *expr*	Returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.
expr { **+**, **–** } *expr*	Addition or subtraction of integer-valued arguments.
expr { *****, **/**, **%** } *expr*	Multiplication, division, or remainder of the integer-valued arguments.
expr **:** *expr*	The matching operator **:** compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are ‘‘anchored’’ (that is, begin with ˆ) and, therefore, ˆ is not a special character, in that context. Normally, the matching operator returns the number of bytes matched (**0** on failure). Alternatively, the **\\(...\\)** pattern symbols can be used to return a portion of the first argument.

EXAMPLES

1. Add 1 to the variable **a**:

```
example% ‘fmlexpr $a + 1 | set -l a‘
```

2. For $a equal to either "/usr/abc/file" or just "file":

```
example% fmlexpr $a : .*/\(.*\) \| $a
```

returns the last segment of a path name (that is, *file*). Watch out for **/** alone as an argument: **fmlexpr** will take it as the division operator (see **NOTES** below).

3. A better representation of example 2.

```
example% fmlexpr //$a : .*/\(.*\)
```

The addition of the **//** characters eliminates any ambiguity about the division operator (because it makes it impossible for the left-hand expression to be interpreted as the division operator), and simplifies the whole expression.

4. Return the number of characters in **$VAR**.

```
example% fmlexpr $VAR : .*
```

SEE ALSO

ed(1), **expr**(1), **set**(1F), **sh**(1)

DIAGNOSTICS

As a side effect of expression evaluation, **fmlexpr** returns the following exit values:

0	if the expression is neither null nor **0** (that is, TRUE)
1	if the expression *is* null or **0** (that is, FALSE)
2	for invalid expressions (that is, FALSE).

syntax error
for operator / operand errors

non-numeric argument
if arithmetic is attempted on such a string

In the case of syntax errors and non-numeric arguments, an error message will be printed at the current cursor position. Use **refresh** to redraw the screen.

NOTES

After argument processing by FMLI, **fmlexpr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an **=**, the command:

```
example% fmlexpr $a = =
```

looks like:

```
example% fmlexpr = = =
```

as the arguments are passed to **fmlexpr** (and they will all be taken as the **=** operator). The following works, and returns TRUE:

```
example% fmlexpr X$a = X=
```

NAME

fmlgrep – search a file for a pattern

SYNOPSIS

fmlgrep [**–b**] [**–c**] [**–i**] [**–l**] [**–n**] [**–s**] [**–v**] *limited_regular_expression* [*filename*. . .]

DESCRIPTION

fmlgrep searches *filename* for a pattern and prints all lines that contain that pattern. The **fmlgrep** function uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with **ed**(1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful when using FMLI special characters (for instance, **$, `, ´,** ") in *limited_regular_expression.* It is safest to enclose the entire *limited_regular_expression* in single quotes ´ **...** ´ .

If *filename* is not specified, **fmlgrep** assumes standard input. Normally, each line matched is copied to standard output. The file name is printed before each line matched if there is more than one input-file.

OPTIONS

–b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
–c	Print only a count of the lines that contain the pattern.
–i	Ignore upper / lower case distinction during comparisons.
–l	Print only the names of files with matching lines, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
–n	Precede each line by its line number in the file (first line is 1).
–s	Suppress error messages about nonexistent or unreadable files.
–v	Print all lines except those that contain the pattern.

SEE ALSO

ed(1), **egrep**(1), **fgrep**(1), **fmlcut**(1F), **grep**(1)

DIAGNOSTICS

fmlgrep returns the following exit values:

0 if the pattern is found (that is, TRUE)
1 if the pattern is not found (that is, FALSE)
2 if an invalid expression was used or *filename* is inaccessible

NOTES

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

If there is a line with embedded nulls, **fmlgrep** will only match up to the first null; if it matches, it will print the entire line.

NAME

fmli – invoke FMLI

SYNOPSIS

fmli [–**a** *alias_file*] [–**c** *command_file*] [–**i** *initialization_file*] *filename* . . .

AVAILABILITY

SUNWcsu

DESCRIPTION

The **fmli** command invokes the Form and Menu Language Interpreter and opens the frame(s) specified by the *filename* argument. The *filename* argument is the pathname of the initial frame definition file(s), and must follow the naming convention **Menu.***xxx*, **Form.***xxx* or **Text.***xxx* for a menu, form or text frame respectively, where *xxx* is any string that conforms to UNIX system file naming conventions. The FMLI descriptor **lifetime** will be ignored for all frames opened by argument to **fmli**. These frames have a lifetime of **immortal** by default.

OPTIONS

–**a** *alias_file*

If –**a** is specified, *alias_file* is the name of a file which contains lines of the form *alias*=*pathname*. Thereafter, **$***alias* can be used in definition files to simplify references to objects or devices with lengthy pathnames, or to define a search path (similar to **$PATH** in the UNIX system shell).

–**c** *command_file*

If –**c** is specified, *command_file* is the name of a file in which default FMLI commands can be disabled, and new application-specific commands can be defined. The contents of *command_file* are reflected in the FMLI Command Menu.

–**i** *initialization_file*

If –**i** is specified, *initialization_file* is the name of a file in which the following characteristics of the application as a whole can be specified:

- A transient introductory frame displaying product information
- A banner, its position, and other elements of the banner line
- Color attributes for all elements of the screen
- Screen Labeled Keys (SLKs) and their layout on the screen.

EXAMPLES

To invoke **fmli**:

```
example% fmli Menu.start
```

where **Menu.start** is an example of *filename* named according to the file name conventions for menu definition files explained above.

To invoke **fmli** and name an initialization file:

```
example% fmli -i init.myapp Menu.start
```

where **init.myapp** is an example of *initialization_file*.

ENVIRONMENT

Variables

LOADPFK	Leaving this environment variable unset tells FMLI, for certain terminals like the AT&T 5620 and 630, to download its equivalent character sequences for using function keys into the terminal's programmable function keys, wiping out any settings the user may already have set in the function keys. Setting **LOADPFK=NO** in the environment will prevent this downloading.
COLUMNS	Can be used to override the width of the logical screen defined for the terminal set in **TERM**. For terminals with a 132-column mode, for example, invoking FMLI with the line **COLUMNS=132 fmli** *frame-file* will allow this wider screen width to be used.
LINES	Can be used to override the length of the logical screen defined for the terminal set in **TERM**.

FILES

/usr/bin/fmli

SEE ALSO

vsig(1F)

DIAGNOSTICS

If *filename* is not supplied to the **fmli** command, **fmli** returns the message:

Initial object must be specified.

If *filename* does not exist or is not readable, **fmli** returns an error message and exits. The example command line above returns the following message and exits:

Can't open object "Menu.start"

If *filename* exists, but does not start with one of the three correct object names (**Menu.**, **Form.**, or **Text.**) or if it is named correctly but does not contain the proper data, **fmli** starts to build the screen by putting out the screen labels for function keys, after which it flashes the message:

I do not recognize that kind of object

and then exits.

NAME

fmt − simple text formatters

SYNOPSIS

fmt [−**c**] [−**s**] [−**w** *width* | −*width*] [*inputfile*. . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

fmt is a simple text formatter that fills and joins lines to produce output lines of (up to) the number of characters specified in the −**w** *width* option. The default *width* is 72. **fmt** concatenates the *inputfile*s listed as arguments. If none are given, **fmt** formats text from the standard input.

Blank lines are preserved in the output, as is the spacing between words. **fmt** does not fill lines beginning with a '**.**' (dot), for compatibility with **nroff**(1). Nor does it fill lines starting with "**From:**".

Indentation is preserved in the output, and input lines with differing indentation are not joined (unless −**c** is used).

fmt can also be used as an in-line text filter for **vi**(1); the **vi** command:

```
!}fmt
```

reformats the text between the cursor location and the end of the paragraph.

OPTIONS

−**c**	Crown margin mode. Preserve the indentation of the first two lines within a paragraph, and align the left margin of each subsequent line with that of the second line. This is useful for tagged paragraphs.
−**s**	Split lines only. Do not join short lines to form longer ones. This prevents sample lines of code, and other such formatted text, from being unduly combined.
−**w** *width* | −*width*	Fill output lines to up to *width* columns.

ENVIRONMENT

If any of the **LC_∗** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **fmt** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_∗** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **fmt** behaves.

LC_CTYPE

Determines how **fmt** handles characters. When **LC_CTYPE** is set to a valid value, **fmt** can display and handle text and filenames containing valid characters for that locale. **fmt** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **fmt** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

SEE ALSO **nroff**(1), **vi**(1)

NOTES The −*width* option is acceptable for BSD compatibility, but it may go away in future releases.

NAME

fmtmsg – display a message on stderr or system console

SYNOPSIS

fmtmsg [**–c** *class*] [**–u** *subclass*] [**–l** *label*] [**–s** *severity*] [**–t** *tag*] [**–a** *action*] *text*

AVAILABILITY

SUNWcsu

DESCRIPTION

Based on a message's classification component, **fmtmsg** either writes a formatted message to **stderr** or writes a formatted message to the console.

A formatted message consists of up to five standard components as defined below. The classification and subclass components are not displayed as part of the standard message, but rather define the source of the message and direct the display of the formatted message.

OPTIONS

–c *class* Describes the source of the message. Valid keywords are:

hard	The source of the condition is hardware.
soft	The source of the condition is software.
firm	The source of the condition is firmware.

–u *subclass* A list of keywords (separated by commas) that further defines the message and directs the display of the message. Valid keywords are:

appl	The condition originated in an application. This keyword should not be used in combination with either **util** or **opsys**.
util	The condition originated in a utility. This keyword should not be used in combination with either **appl** or **opsys**.
opsys	The message originated in the kernel. This keyword should not be used in combination with either **appl** or **util**.
recov	The application will recover from the condition. This keyword should not be used in combination with **nrecov**.
nrecov	The application will not recover from the condition. This keyword should not be used in combination with **recov**.
print	Print the message to the standard error stream **stderr**.
console	Write the message to the system console. **print**, **console**, or both may be used.

–l *label* Identifies the source of the message.

–s *severity* Indicates the seriousness of the error. The keywords and definitions of the standard levels of *severity* are:

halt	The application has encountered a severe fault and is halting.
error	The application has detected a fault.
warn	The application has detected a condition that is out of the ordinary and might be a problem.
info	The application is providing information about a condition that is not in error.

–t *tag* The string containing an identifier for the message.

–a *action*	A text string describing the first step in the error recovery process. This string must be written so that the entire *action* argument is interpreted as a single argument. **fmtmsg** precedes each action string with the **TO FIX:** prefix.
text	A text string describing the condition. Must be written so that the entire *text* argument is interpreted as a single argument.

ENVIRONMENT

The environment variables **MSGVERB** and **SEV_LEVEL** control the behavior of **fmtmsg**. **MSGVERB** is set by the administrator in the **/etc/profile** for the system. Users can override the value of **MSGVERB** set by the system by resetting **MSGVERB** in their own **.profile** files or by changing the value in their current shell session. **SEV_LEVEL** can be used in shell scripts.

MSGVERB tells **fmtmsg** which message components to select when writing messages to **stderr**. The value of **MSGVERB** is a colon separated list of optional keywords. **MSGVERB** can be set as follows:

MSGVERB=[*keyword*[**:***keyword*[**:**...]]]
export MSGVERB

Valid *keywords* are: **label**, **severity**, **text**, **action**, and **tag**. If **MSGVERB** contains a keyword for a component and the component's value is not the component's null value, **fmtmsg** includes that component in the message when writing the message to **stderr**. If **MSGVERB** does not include a keyword for a message component, that component is not included in the display of the message. The keywords may appear in any order. If **MSGVERB** is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, **fmtmsg** selects all components.

MSGVERB affects only which message components are selected for display. All message components are included in console messages.

SEV_LEVEL defines severity levels and associates print strings with them for use by **fmtmsg**. The standard severity levels shown below cannot be modified. Additional severity levels can be defined, redefined, and removed.

0	(no severity is used)
1	HALT
2	ERROR
3	WARNING
4	INFO

SEV_LEVEL is set as follows:

SEV_LEVEL=[*description*[**:***description*[**:**...]]]
export SEV_LEVEL

description is a comma-separated list containing three fields:

description=*severity_keyword***,***level***,***printstring* *severity_keyword* is a character string used as the keyword with the **–s** *severity* option to **fmtmsg**.

level is a character string that evaluates to a positive integer (other than **0**, **1**, **2**, **3**, or **4**, which are reserved for the standard severity levels). If the keyword *severity_keyword* is used, *level* is the severity value passed on to **fmtmsg**(3C).

printstring is the character string used by **fmtmsg** in the standard message format whenever the severity value *level* is used.

If **SEV_LEVEL** is not defined, or if its value is null, no severity levels other than the defaults are available. If a *description* in the colon separated list is not a comma separated list containing three fields, or if the second field of a comma separated list does not evaluate to a positive integer, that *description* in the colon separated list is ignored.

EXAMPLES

Example 1: The following example of **fmtmsg** produces a complete message in the standard message format and displays it to the standard error stream:

```
example% fmtmsg –c soft –u recov,print,appl –l UX:cat –s error -t UX:cat:001
–a "refer to manual" "invalid syntax"
```

produces:

```
UX:cat: ERROR: invalid syntax
TO FIX: refer to manual   UX:cat:138
```

Example 2: When the environment variable **MSGVERB** is set as follows:

```
MSGVERB=severity:text:action
```

and Example 1 is used, **fmtmsg** produces:

```
ERROR: invalid syntax
TO FIX: refer to manual
```

Example 3: When the environment variable **SEV_LEVEL** is set as follows:

```
SEV_LEVEL=note,5,NOTE
```

the following **fmtmsg** command:

```
example% fmtmsg –c soft –u print –l UX:cat –s note –a "refer to manual"
"invalid syntax"
```

produces:

```
  NOTE: invalid syntax
To FIX: refer to manual
```

and displays the message on **stderr**.

SEE ALSO

addseverity(3C), **fmtmsg**(3C)

DIAGNOSTICS

The exit codes for **fmtmsg** are the following:

0	All the requested functions were executed successfully.
1	The command contains a syntax error, an invalid option, or an invalid argument to an option.
2	The function executed with partial success, however the message was not displayed on **stderr**.
4	The function executed with partial success, however the message was not displayed on the system console.
32	No requested functions were executed successfully.

NAME

fold – fold long lines

SYNOPSIS

fold [–**w** *width* | –*width*] [*filename* . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

Fold the contents of the specified *filename*s, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *width* should be a multiple of 8 if tabs are present, or the tabs should be expanded.

OPTIONS

–**w** *width* | –*width*

The contents of the specified *filenames* or the standard input are folded breaking the lines to have maximum width *width*, given either of these options.

SEE ALSO

pr(1)

NOTES

Folding may not work correctly if underlining is present.

The –*width* option is provided as a transition tool only. It will be removed in future releases.

NAME

for, foreach – shell built-in functions to repeatedly execute action(s) for a selected number of times

SYNOPSIS

sh

for *name* [**in** *word*. . .] **;** **do** *actions* **;** **done**

csh

foreach *var* **(***wordlist***)**
. . .
end

ksh

for *identifier* [**in** *word* . . .] **;** **do** *actions* **;** **done**

DESCRIPTION

sh

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* **. . .** is omitted, then the **for** command executes the **do** *actions* once for each positional parameter that is set. Execution ends when there are no more words in the list.

csh

The variable *var* is successively set to each member of *wordlist*. The sequence of commands between this command and the matching **end** is executed for each new value of *var*. Both **foreach** and **end** must appear alone on separate lines.

The built-in command **continue** may be used to terminate the execution of the current iteration of the loop and the built-in command **break** may be used to terminate execution of the **foreach** command. When this command is read from the terminal, the loop is read once prompting with **?** before any statements in the loop are executed.

ksh

Each time a **for** command is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* **. . .** is omitted, then the **for** command executes the **do** *actions* once for each positional parameter that is set Execution ends when there are no more words in the list.

SEE ALSO

break(1), **csh**(1), **ksh**(1), **sh**(1)

NOTES

Both the Bourne shell, **sh**, and the Korn shell, **ksh**, can use the semicolon and the carriage return interchangeably in their syntax of the **if**, **for**, and **while** built-in commands.

NAME

from – display the sender and date of newly-arrived mail messages

SYNOPSIS

/usr/ucb/from [**–s** *sender*] [*username*]

AVAILABILITY

SUNWscpu

DESCRIPTION

from prints out the mail header lines in your mailbox file to show you who your mail is from. If *username* is specified, then *username*'s mailbox is examined instead of your own.

OPTIONS

–s *sender*	Only display headers for mail sent by *sender*.

FILES

/var/spool/mail/∗

SEE ALSO

biff(1B), **mail**(1B)

NAME

ftp – file transfer program

SYNOPSIS

ftp [**–dgintv**] [*hostname*]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **ftp** command is the user interface to the Internet standard File Transfer Protocol (FTP). **ftp** transfers files to and from a remote network site.

The client host with which **ftp** is to communicate may be specified on the command line. If this is done, **ftp** immediately attempts to establish a connection to an FTP server on that host; otherwise, **ftp** enters its command interpreter and awaits instructions from the user. When **ftp** is awaiting commands from the user, it displays the prompt **ftp>**.

OPTIONS

The following options may be specified at the command line, or to the command interpreter:

–d	Enable debugging.
–g	Disable filename “globbing.”
–i	Turn off interactive prompting during multiple file transfers.
–n	Do not attempt “auto-login” upon initial connection. If auto-login is not disabled, **ftp** checks the **.netrc** file in the user’s home directory for an entry describing an account on the remote machine. If no entry exists, **ftp** will prompt for the login name of the account on the remote machine (the default is the login name on the local machine), and, if necessary, prompts for a password and an account with which to login.
–t	Enable packet tracing (unimplemented).
–v	Show all responses from the remote server, as well as report on data transfer statistics. This is turned on by default if **ftp** is running interactively with its input coming from the user’s terminal.

The following commands can be specified to the command interpreter:

! [*command*]

Run *command* as a shell command on the local machine. If no *command* is given, invoke an interactive shell.

$ *macro-name* [*args*]

Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

account [*passwd*]

Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is not specified, the local file name is used, subject to alteration by any **ntrans** or **nmap** settings. File transfer uses the current settings for "representation type", "file structure", and "transfer mode".

ascii Set the "representation type" to "network ASCII". This is the default type.

bell Sound a bell after each file transfer command is completed.

binary Set the "representation type" to "image".

bye Terminate the FTP session with the remote server and exit **ftp**. An EOF will also terminate the session and exit.

case Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle RETURN stripping during "network ASCII" type file retrieval. Records are denoted by a RETURN / LINEFEED sequence during "network ASCII" type file transfer. When **cr** is on (the default), RETURN characters are stripped from this sequence to conform with the UNIX system single LINEFEED record delimiter. Records on non-UNIX-system remote hosts may contain single LINEFEED characters; when an "network ASCII" type transfer is made, these LINEFEED characters may be distinguished from a record delimiter only when **cr** is off.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug
Toggle debugging mode. When debugging is on, **ftp** prints each command sent to the remote machine, preceded by the string ––>.

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is –, output is sent to the terminal.

disconnect
A synonym for **close**.

form [*format-name*]

Set the carriage control format subtype of the “representation type” to *format-name*. The only valid *format-name* is **non-print**, which corresponds to the default “non-print” subtype.

get *remote-file* [*local-file*]

Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current **case**, **ntrans**, and **nmap** settings. The current settings for “representation type”, “file structure”, and “transfer mode” are used while transferring the file.

glob Toggle filename expansion, or “globbing”, for **mdelete**, **mget** and **mput**. If globbing is turned off, filenames are taken literally.

Globbing for **mput** is done as in **sh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine, and the lists are not merged.

Expansion of a directory name is likely to be radically different from expansion of the name of an ordinary file: the exact result depends on the remote operating system and FTP server, and can be previewed by doing **mls** *remote-files* –.

mget and **mput** are not meant to transfer entire directory subtrees of files. You can do this by transferring a **tar**(1) archive of the subtree (using a “representation type” of “image” as set by the **binary** command).

hash Toggle hash-sign (**#**) printing for each data block transferred. The size of a data block is 8192 bytes.

help [*command*]

Print an informative message about the meaning of *command*. If no argument is given, **ftp** prints a list of the known commands.

lcd [*directory*]

Change the working directory on the local machine. If no *directory* is specified, the user’s home directory is used.

ls [*remote-directory* | **-al**] [*local-file*]

Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used.

The **-a** option lists all entries, including those that begin with a dot (**.**), which are normally not listed. The **-l** option lists files in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file. If the file is a special file, the size field instead contains the major and minor device numbers rather than a size. If the file is a symbolic link, the filename is printed followed by “→” and the pathname of the referenced file.

If no local file is specified, or if *local-file* is –, the output is sent to the terminal.

macdef *macro-name*

Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive NEWLINE characters in a file or RETURN characters from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed.

The macro processor interprets **$** and \ as special characters. A **$** followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A **$** followed by an **i** signals that macro processor that the executing macro is to be looped. On the first pass **$i** is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A \ followed by any character is replaced by that character. Use the \ to prevent special treatment of the **$**.

mdelete *remote-files*

Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*

Like **dir**, except multiple remote files may be specified. If interactive prompting is on, **ftp** will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*

Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which can be changed with **lcd** *directory*; new local directories can be created with **! mkdir** *directory*.

mkdir *directory-name*

Make a directory on the remote machine.

mls *remote-files local-file*

Like **ls**(1), except multiple remote files may be specified. If interactive prompting is on, **ftp** will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]

Set the “transfer mode” to *mode-name*. The only valid *mode-name* is **stream**, which corresponds to the default “stream” mode. This implementation only supports **stream**, and requires that it be specified.

mput *local-files*

Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

nmap [*inpattern outpattern*]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences **$1**, **$2**, . . . , **$9** in *inpattern*. Use \ to prevent this special treatment of the **$** character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values.

For example, given *inpattern* **$1.$2** and the remote file name **mydata.data**, **$1** would have the value **mydata**, and **$2** would have the value **data**.

The *outpattern* determines the resulting mapped filename. The sequences **$1**, **$2**, . . . , **$9** are replaced by any value resulting from the *inpattern* template. The sequence **$0** is replaced by the original filename. Additionally, the sequence [*seq1* , *seq2*] is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*.

For example, the command **nmap $1.$2.$3 [$1,$2].[$2,file]** would yield the output filename **myfile.data** for input filenames **myfile.data** and **myfile.data.old**, **myfile.file** for the input filename **myfile**, and **myfile.myfile** for the input filename **.myfile**. SPACE characters may be included in *outpattern*, as in the example **nmap $1 | sed** "**s/ ∗$//**" **> $1**. Use the \ character to prevent special treatment of the **$**, **[**, **]**, and **,**, characters.

ntrans [*inchars* [*outchars*]]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename, and characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

Only 16 characters can be translated when using the **ntrans** command under **ftp**. Use **case** (described above) if needing to convert the entire alphabet.

open *host* [*port*]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, **ftp** will attempt to contact an FTP server at that port. If the *auto-login* option is on (default setting), **ftp** will also attempt to automatically log the user in to the FTP server.

prompt

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. By default, prompting is turned on. If prompting is turned off, any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

proxy *ftp-command*

Execute an FTP command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command **proxy ?** to see other FTP commands executable on the secondary connection.

The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mputd**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection.

Third party file transfers depend upon support of the **PASV** command by the server on the secondary control connection.

put *local-file* [*remote-file*]

Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any **ntrans** or **nmap** settings in naming the remote file. File transfer uses the current settings for “representation type”, “file structure”, and “transfer mode”.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 . . .*

Send the arguments specified, verbatim, to the remote FTP server. A single FTP reply code is expected in return. (The **remotehelp** command displays a list of valid arguments.)

quote should be used only by experienced users who are familiar with the FTP protocol.

recv *remote-file* [*local-file*]

A synonym for **get**.

remotehelp [*command-name*]

Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

rename *from to*
Rename the file *from* on the remote machine to have the name *to*.

reset Clear reply queue. This command re-synchronizes command / reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

rmdir *directory-name*
Delete a directory on the remote machine.

runique
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a **.1** is appended to the name. If the resulting name matches another existing file, a **.2** is appended to the original name. If this process continues up to **.99**, an error message is printed, and the transfer does not take place. The generated unique filename will be reported. **runique** will not affect local files generated from a shell command. The default value is off.

send *local-file* [*remote-file*]
A synonym for **put**.

sendport
Toggle the use of **PORT** commands. By default, **ftp** will attempt to use a **PORT** command when establishing a connection for each data transfer. The use of **PORT** commands can prevent delays when performing multiple file transfers. If the **PORT** command fails, **ftp** will use the default data port. When the use of **PORT** commands is disabled, no attempt will be made to use **PORT** commands for each data transfer. This is useful when connected to certain FTP implementations that ignore **PORT** commands but incorrectly indicate they have been accepted.

status Show the current status of **ftp**.

struct [*struct-name*]
Set the file structure to *struct-name*. The only valid *struct-name* is **file**, which corresponds to the default “file” structure. The implementation only supports **file**, and requires that it be specified.

sunique
Toggle storing of files on remote machine under unique file names. The remote FTP server must support the **STOU** command for successful completion. The remote server will report the unique name. Default value is off.

tenex Set the “representation type” to that needed to talk to TENEX machines.

trace Toggle packet tracing (unimplemented).

type [*type-name*]
Set the “representation type” to *type-name*. The valid *type-name*s are **ascii** for “network ASCII”, **binary** or **image** for “image”, and **tenex** for “local byte size” with a byte size of 8 (used to talk to TENEX machines). If no type is specified, the current type is printed. The default type is “network ASCII”.

user *user-name* [*password*] [*account*]

Identify yourself to the remote FTP server. If the password is not specified and the server requires it, **ftp** will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless **ftp** is invoked with “auto-login” disabled, this process is done automatically on initial connection to the FTP server.

verbose

Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose mode is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose mode is on if **ftp**’s commands are coming from a terminal, and off otherwise.

? [*command*]

A synonym for **help**.

Command arguments which have embedded spaces may be quoted with quote (") marks.

If any command argument which is not indicated as being optional is not specified, **ftp** will prompt for that argument.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key. Sending transfers will be immediately halted. Receiving transfers will be halted by sending an FTP protocol **ABOR** command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server’s support for **ABOR** processing. If the remote server does not support the **ABOR** command, an **ftp>** prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when **ftp** has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the **ABOR** processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local **ftp** program must be killed by hand.

FILE NAMING CONVENTIONS

Local files specified as arguments to **ftp** commands are processed according to the following rules.

1) If the file name – is specified, the standard input (for reading) or standard output (for writing) is used.

2) If the first character of the file name is **|**, the remainder of the argument is interpreted as a shell command. **ftp** then forks a shell, using **popen**(3S) with the argument supplied, and reads (writes) from the standard output (standard input) of that shell. If the shell command includes SPACE characters, the argument must be quoted; for example "**| ls** –**lt**". A particularly useful example of this mechanism is: "**dir | more**".

3) Failing the above checks, if globbing is enabled, local file names are expanded according to the rules used in the **sh**(1); see the **glob** command. If the **ftp** command expects a single local file (for example, **put**), only the first filename generated by the globbing operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer.

The “representation type” may be one of “network ASCII”, “EBCDIC”, “image”, or “local byte size” with a specified byte size (for PDP-10’s and PDP-20’s mostly). The “network ASCII” and “EBCDIC” types have a further subtype which specifies whether vertical format control (NEWLINE characters, form feeds, etc.) are to be passed through (“non-print”), provided in TELNET format (“TELNET format controls”), or provided in ASA (FORTRAN) (“carriage control (ASA)”) format. **ftp** supports the “network ASCII” (subtype “non-print” only) and “image” types, plus “local byte size” with a byte size of 8 for communicating with TENEX machines.

The “file structure” may be one of **file** (no record structure), **record**, or **page**. **ftp** supports only the default value, which is **file**.

The “transfer mode” may be one of **stream**, **block**, or **compressed**. **ftp** supports only the default value, which is **stream**.

FILES

˜/.netrc

SEE ALSO

ls(1), **rcp**(1), **sh**(1), **tar**(1), **ftpd**(1M), **popen**(3S), **netrc**(4)

NOTES

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2 BSD code handling transfers with a “representation type” of “network ASCII” has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using a “representation type” of “network ASCII”. Avoid this problem by using the “image” type.

NAME

function – shell built-in command to define a function which is usable within this shell

SYNOPSIS

ksh

function *identifier* **{** *list* **;}**
*identifier***( ) {** *list* **;}**

DESCRIPTION

ksh

function defines a function which is referenced by *identifier*. The body of the function is the *list* of commands between **{** and **}**.

Alternatively, omitting the **function** keyword and appending the *identifier* with a set of enclosed parentheses will accomplish the same function definition.

SEE ALSO

ksh(1)

NAME

gcore – get core images of running processes

SYNOPSIS

gcore [**−o** *filename*] [**−p** *procdir*] *process-id* . . .

DESCRIPTION

gcore creates a core image of each specified process. The name of the core image file for the process whose process ID is *process-id* will be **core.***process-id*.

OPTIONS

−o Substitutes *filename* in place of **core** as the first part of the name of the core image files.

FILES

core.process-id	core images

SEE ALSO

csh(1), **kill**(1), **ptrace**(2)

NAME

gencat – generate a formatted message catalog

SYNOPSIS

gencat *catfile msgfile*. . .

AVAILABILITY

SUNWloc

DESCRIPTION

gencat merges the message text source file(s) *msgfile* into a formatted message database *catfile.* The database *catfile* is created if it does not already exist. If *catfile* does exist, its messages are included in the new *catfile.* If set and message numbers collide, the new message-text defined in *msgfile* replaces the old message text currently contained in *catfile.* The message text source file (or set of files) input to **gencat** can contain either set and message numbers or simply message numbers, in which case the set **NL_SETD** (see **nl_types**(5)) is assumed.

The format of a message text source file is defined as follows. Note that the fields of a message text source line are separated by a single ASCII space or tab character. Any other ASCII spaces or tabs are considered as part of the subsequent field.

$set n comment

Where *n* specifies the set identifier of the following messages until the next **$set**, **$delset**, or end-of-file appears. *n* must be a number in the range (1–{**NL_SETMAX**}). Set identifiers within a single source file need not be contiguous. Any string following the set identifier is treated as a comment. If no **$set** directive is specified in a message text source file, all messages are located in the default message set **NL_SETD**.

$delset n comment

Deletes message set *n* from an existing message catalog. Any string following the set number is treated as a comment. (Note: if *n* is not a valid set it is ignored.)

$ comment

A line beginning with a dollar symbol $ followed by an ASCII space or tab character is treated as a comment.

m message-text

The *m* denotes the message identifier, a number in the range (1-{**NL_MSGMAX**}). The message-text is stored in the message catalog with the set identifier specified by the last **$set** directive, and with message identifier *m.* If the message-text is empty, and an ASCII space or tab field separator is present, an empty string is stored in the message catalog. If a message source line has a message number, but neither a field separator nor message-text , the existing message with that number (if any) is deleted from the catalog. Message identifiers need not be contiguous. The length of message-text must be in the range (0–{**NL_TEXTMAX**}).

$quote c

This line specifies an optional quote character *c*, which can be used to surround message-text so that trailing spaces or null (empty) messages are visible in a message source line. By default, or if an empty **$quote** directive is supplied, no quoting of message-text will be recognized.

Empty lines in a message text source file are ignored.

Text strings can contain the special characters and escape sequences defined in the following table:

Description	Symbol	Sequence
newline	NL(LF)	\n
horizontal tab	HT	\t
vertical tab	VT	\v
backspace	BS	\b
carriage return	CR	\r
form feed	FF	\f
backslash	\	\\
bit pattern	ddd	\ddd

The escape sequence **\ddd** consists of backslash followed by 1, 2 or 3 octal digits, which are taken to specify the value of the desired character. If the character following a backslash is not one of those specified, the backslash is ignored.

Backslash followed by an ASCII newline character is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

```
1 This line continues \
to the next line
```

which is equivalent to:

```
1 This line continues to the next line
```

ENVIRONMENT

After message catalogs are generated and installed, the **LANG** environment variable, or the **LC_MESSAGES** environment if present, determines the prevailing message locale.

SEE ALSO

mkmsgs(1), **catgets**(3C), **catopen**(3C), **gettxt**(3C), **environ**(5), **nl_types**(5)

NAME

getfrm – returns the current frameID number

SYNOPSIS

getfrm

DESCRIPTION

getfrm returns the current frameID number. The frameID number is a number assigned to the frame by FMLI and displayed flush left in the frame's title bar. If a frame is closed its frameID number may be reused when a new frame is opened. **getfrm** takes no arguments.

EXAMPLES

If a menu whose frameID is **3** defines an item to have this **action** descriptor:

```
action=open text stdtext `getfrm`
```

the text frame defined in the definition file **stdtext** would be passed the argument **3** when it is opened.

NOTES

It is not a good idea to use **getfrm** in a backquoted expression coded on a line by itself. Stand-alone backquoted expressions are evaluated before any descriptors are parsed, thus the frame is not yet fully current, and may not have been assigned a frameID number.

NAME

getitems – returns a list of currently marked menu items

SYNOPSIS

getitems [*delimiter_string*]

DESCRIPTION

The **getitems** function returns the value of **lininfo** if defined, else it returns the value of the **name** descriptor, for all currently marked menu items. Each value in the list is delimited by *delimiter_string.* The default value of *delimiter_string* is newline.

EXAMPLES

The **done** descriptor in the following menu definition file executes **getitems** when the user presses ENTER (note that the menu is multiselect):

```
Menu="Example"
multiselect=TRUE
done=`getitems ":" | message`

name="Item 1"
action=`message "You selected item 1"`

name="Item 2"
lininfo="This is item 2"
action=`message "You selected item 2"`

name="Item 3"
action=`message "You selected item 3"`
```

If a user marked all three items in this menu, pressing ENTER would cause the following string to be displayed on the message line:

```
Item 1:This is item 2:Item 3
```

Note: Because **lininfo** is defined for the second menu item, its value is displayed instead of the value of the **name** descriptor.

NAME

getopt – parse command options

SYNOPSIS

set — `getopt** *optstring* **$*`

AVAILABILITY

SUNWcsu

DESCRIPTION

The **getopts** command supersedes **getopt**. For more information, see the NOTES below.

getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *optstring* is a string of recognized option letters; see **getopt**(3C). If a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option — is used to delimit the end of the options. If it is used explicitly, **getopt** recognizes it; otherwise, **getopt** generates it; in either case, **getopt** places it at the end of the options. The positional parameters (**$1 $2** . . .) of the shell are reset so that each option is preceded by a – and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLES

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set — `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        –a| –b)         FLAG=$i; shift;;
        –o)             OARG=$2; shift 2;;
        —)              shift; break;;
        esac
done
```

This code accepts any of the following as equivalent:

```
cmd –aoarg filename1 filename2
cmd –a –o arg filename1 filename2
cmd –oarg –a filename1 filename2
cmd –a –oarg — filename1 filename2
```

SEE ALSO

shell_builtins(1), **sh**(1), **getopt**(3C)

DIAGNOSTICS

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NOTES

getopt will not be supported in the next major release. For this release a conversion tool has been provided, **getoptcvt**. For more information about **getopts** and **getoptcvt**, see **getopts**(1).

Reset **optind** to 1 when rescanning the options.

getopt does not support the part of Rule 8 of the command syntax standard (see **intro**(1)) that permits groups of option-arguments following an option to be separated by white space and quoted. For example,

```
cmd –a –b –o "xxx z yy" filename
```

is not handled correctly. To correct this deficiency, use the **getopts** command in place of **getopt**.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring* (referring to the earlier EXAMPLE section, but using the following command line: **cmd -o -a filename**), **getopt** always treats **–a** as an option-argument to **–o**; it never recognizes **–a** as an option. For this case, the **for** loop in the example shifts past the *filename* argument.

NAME

getoptcvt – convert to getopts to parse command options

SYNOPSIS

/usr/lib/getoptcvt [**–b**] *filename*

/usr/lib/getoptcvt

DESCRIPTION

/usr/lib/getoptcvt reads the shell script in *filename*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

getopts is a built-in Bourne shell command used to parse positional parameters and to check for valid options. See **sh**(1). It supports all applicable rules of the command syntax standard (see Rules 3-10, **intro**(1)). It should be used in place of the **getopt** command. (See the **NOTES** section below.) The syntax for the shell's built-in **getopts** command is:

getopts *optstring name* [*argument* . . .]

optstring must contain the option letters the command using **getopts** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** places the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell script is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, **getopts** exits with a non-zero exit status. The special option –– may be used to delimit the end of the options.

By default, **getopts** parses the positional parameters. If extra arguments (*argument* . . .) are given on the **getopts** command line, **getopts** parses them instead.

So that all new commands will adhere to the command syntax standard described in **intro**(1), they should use **getopts** or **getopt** to parse positional parameters and check for options that are valid for that command (see the NOTES section below).

OPTIONS

–b Make the converted script portable to earlier releases of the UNIX system. **/usr/lib/getoptcvt** modifies the shell script in *filename* so that when the resulting shell script is executed, it determines at run time whether to invoke **getopts** or **getopt**.

EXAMPLES

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
done
shift `expr $OPTIND − 1`
```

This code accepts any of the following as equivalent:

```
cmd −a −b −o "xxx z yy" filename
cmd −a −b −o "xxx z yy" −− filename
cmd −ab −o xxx,z,yy filename
cmd −ab −o "xxx z yy" filename
cmd −o xxx,z,yy −b −a filename
```

SEE ALSO

intro(1), **sh**(1), **shell_builtins**(1), **getopt**(3C)

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NOTES

Although the following command syntax rule (see **intro**(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLES** section above, **a** and **b** are options, and the option **o** requires an option-argument. The following example violates Rule 5: options with option-arguments must not be grouped with other options:

```
example% cmd −aboxxx filename
```

The following example violates Rule 6: there must be white space after an option that takes an option-argument:

```
example% cmd −ab −oxxx filename
```

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

NAME

getopts – shell built-in function to parse command-line options

SYNOPSIS

sh

getopts *optstring name* [*argument* . . .]

ksh

getopts *optstring name* [*arg* . . .]

DESCRIPTION

sh

getopts is a built-in Bourne shell command used to parse positional parameters and to check for valid options. See **sh**(1). It supports all applicable rules of the command syntax standard (see Rules 3-10, **intro**(1)). It should be used in place of the **getopt** command.

optstring must contain the option letters the command using **getopts** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** places the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell script is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, **getopts** exits with a non-zero exit status. The special option –– may be used to delimit the end of the options.

By default, **getopts** parses the positional parameters. If extra arguments (*argument* . . .) are given on the **getopts** command line, **getopts** parses them instead.

/usr/lib/getoptcvt reads the shell script in *filename*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

So that all new commands will adhere to the command syntax standard described in **intro**(1), they should use **getopts** or **getopt** to parse positional parameters and check for options that are valid for that command.

Examples:

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
     case $c in
     a | b)     FLAG=$c;;
     o)     OARG=$OPTARG;;
     \?)    echo $USAGE
            exit 2;;
     esac
done
```

```
shift `expr $OPTIND –1`
```

This code accepts any of the following as equivalent:

```
cmd –a –b –o "xxx z yy" filename
cmd –a –b –o "xxx z yy" –– filename
cmd –ab –o xxx,z,yy filename
cmd –ab –o "xxx z yy" filename
cmd –o xxx,z,yy –b –a filename
```

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

Although the following command syntax rule (see **intro**(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the EXAMPLES section above, **a** and **b** are options, and the option **o** requires an option-argument. The following example violates Rule 5: options with option-arguments must not be grouped with other options:

```
example% cmd –aboxxx filename
```

The following example violates Rule 6: there must be white space after an option that takes an option-argument:

```
example% cmd –ab –oxxx filename
```

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

ksh Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a **+** or a –. An option not beginning with **+** or – or the argument – – ends the options. *optstring* contains the letters that **getopts** recognizes. If a letter is followed by a **:**, that option is expected to have an argument. The options can be separated from the argument by blanks.

getopts places the next option letter it finds inside variable *name* each time it is invoked with a **+** prepended when *arg* begins with a **+**. The index of the next *arg* is stored in **OPTIND**. The option argument, if any, gets stored in **OPTARG**.

A leading **:** in *optstring* causes **getopts** to store the letter of an invalid option in **OPTARG**, and to set *name* to **?** for an unknown option and to **:** when a required option is missing. Otherwise, **getopts** prints an error message. The exit status is non-zero when there are no more options.

For a further discussion of the Korn shell's **getopts** built-in command, see the previous discussion in the Bourne shell, **sh**, section of this manpage.

SEE ALSO

getopt(1), **getoptcvt**(1), **intro**(1), **ksh**(1), **sh**(1)

NAME

gettext – retrieve text string from message database

SYNOPSIS

gettext [*textdomain*] *msgid*

AVAILABILITY

SUNWcsu

DESCRIPTION

gettext retrieves a translated text string corresponding to string *msgid* from a message object generated with **msgfmt**(1). The message object name is derived from the optional argument *textdomain* if present, otherwise from the **TEXTDOMAIN** environment. If no domain is specified, or if a corresponding string cannot be found, **gettext** prints *msgid*.

Ordinarily **gettext** looks for its message object in **/usr/lib/locale/***lang***/LC_MESSAGES** where *lang* is the locale name. If present, the **TEXTDOMAINDIR** environment variable replaces the pathname component up to *lang*.

This command interprets C escape sequences such as **\t** for tab. Use \\ to print a backslash. To produce a message on a line of its own, either put a **\n** at the end of *msgid*, or use this command in conjunction with **printf**(1).

ENVIRONMENT

LANG Specifies locale name.

LC_MESSAGES
Specifies messaging locale, and if present overrides **LANG** for messages.

TEXTDOMAIN
Specifies the text domain name, which is identical to the message object filename without **.mo** suffix.

TEXTDOMAINDIR
Specifies the pathname to the message database, and if present replaces **/usr/lib/locale**.

SEE ALSO

msgfmt(1), **printf**(1), **gettext**(3I), **setlocale**(3C)

NOTES

This is the shell equivalent of the library routine **gettext**(3I).

NAME

gettxt – retrieve a text string from a message database

SYNOPSIS

gettxt *msgfile***:***msgnum* **[***dflt_msg***]**

AVAILABILITY

SUNWloc

DESCRIPTION

gettxt retrieves a text string from a message file in the directory **/usr/lib/locale/***locale***/LC_MESSAGES**. The directory name *locale* corresponds to the language in which the text strings are written; see **setlocale**(3C).

msgfile	Name of the file in the directory **/usr/lib/locale/***locale***/LC_MESSAGES** to retrieve *msgnum* from. The name of *msgfile* can be up to 14 characters in length, but may not contain either \0 (null) or the ASCII code for **/** (slash) or **:** (colon).
msgnum	Sequence number of the string to retrieve from *msgfile*. The strings in *msgfile* are numbered sequentially from *1* to *n*, where *n* is the number of strings in the file.
dflt_msg	Default string to be displayed if **gettxt** fails to retrieve *msgnum* from *msgfile*. Nongraphic characters must be represented as alphabetic escape sequences.

The text string to be retrieved is in the file *msgfile*, created by the **mkmsgs**(1) utility and installed under the directory **/usr/lib/locale/***locale***/LC_MESSAGES**. You control which directory is searched by setting the environment variable **LC_MESSAGES**. If **LC_MESSAGES** is not set, the environment variable **LANG** will be used. If **LANG** is not set, the files containing the strings are under the directory **/usr/lib/locale/C/LC_MESSAGES**.

If **gettxt** fails to retrieve a message in the requested language, it will try to retrieve the same message from **/usr/lib/locale/C/LC_MESSAGES/***msgfile.* If this also fails, and if *dflt_msg* is present and non-null, then it will display the value of *dflt_msg*; if *dflt_msg* is not present or is null, then it will display the string **Message not found!!**.

EXAMPLES

If the environment variables **LANG** or **LC_MESSAGES** have not been set to other than their default values, the following example:

```
example% gettxt UX:10 "hello world\n"
```

will try to retrieve the 10th message from **/usr/lib/locale/C/UX/***msgfile.* If the retrieval fails, the message "hello world," followed by a newline, will be displayed.

ENVIRONMENT

If any of the **LC_∗** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **gettxt** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_∗** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **gettxt** behaves.

LC_CTYPE
Determines how **gettxt** handles characters. When **LC_CTYPE** is set to a valid value, **gettxt** can display and handle text and filenames containing valid characters for that locale. **gettxt** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **gettxt** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/usr/lib/locale/C/LC_MESSAGES/*	default message files created by **mkmsgs**(1)
/usr/lib/locale/*locale***/LC_MESSAGES/***	message files for different languages created by **mkmsgs**(1)

SEE ALSO

exstr(1), **mkmsgs**(1), **srchtxt**(1), **gettxt**(3C), **setlocale**(3C), **environ**(5)

NAME

glob – shell built-in function to expand a word list

SYNOPSIS

csh

glob *wordlist*

DESCRIPTION

csh

glob performs filename expansion on *wordlist*. Like **echo**(1), but no \ escapes are recognized. Words are delimited by **NULL** characters in the output.

SEE ALSO

csh(1), **echo**(1)

NAME

gprof – display call-graph profile data

SYNOPSIS

gprof [–**abcCDsz**] [–**e** *function-name*] [–**E** *function-name*] [–**f** *function-name*]
[–**F** *function-name*] [*image-file* [*profile-file* . . .]] [–**n** *number of functions*]

DESCRIPTION

gprof produces an execution profile of a program. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file which is created by programs compiled with the –**xpg** option of **cc**(1B), or –**pg** for other compilers. These options also link in versions of the library routines which are compiled for profiling. The symbol table in the executable image file *image-file* (**a.out** by default) is read and correlated with the call graph profile file *profile-file* (**gmon.out** by default). If more than one profile file is specified, the **gprof** output shows the sum of the profile information in the given profile files.

First, execution times for each routine are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. The first listing shows the functions sorted according to the time they represent, including the time of their call graph descendants. Below each function entry is shown its (direct) call-graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendants is propagated to its (direct) call-graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

Next, a flat profile is given, similar to that provided by **prof**(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time. Finally, an index is given, showing the correspondence between function names and call-graph profile index numbers.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(5)).

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call-graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

The profiled program must call **exit**(2) or return normally for the profiling information to be saved in the **gmon.out** file.

OPTIONS

–**a**	Suppress printing statically declared functions. If this option is given, all relevant information about the static function (for instance, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the **a.out** file.
–**b**	Brief. Suppress descriptions of each field in the profile.
–**C**	Demangle C++ symbol names before printing them out.

–c Discover the static call-graph of the program by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

–D Produce a profile file **gmon.sum** that represents the difference of the profile information in all specified profile files. This summary profile file may be given to subsequent executions of **gprof** (also with –**D**) to summarize profile data across several runs of an **a.out** file. (See also the –**s** option.)

As an example, suppose function A calls function B **n** times in profile file **gmon.sum**, and **m** times in profile file **gmon.out**. With –**D**, a new **gmon.sum** file will be created showing the number of calls from A to B as **n-m**.

–**E** *function-name*
Suppress printing the graph profile entry for routine *function-name* (and its descendants) as –**e**, below, and also exclude the time spent in *function-name* (and its descendants) from the total and percentage time computations. More than one –**E** option may be given. For example:

‘–**E** *mcount* –**E** *mcleanup*’

is the default.

–**e** *function-name*
Suppress printing the graph profile entry for routine *function-name* and all its descendants (unless they have other ancestors that are not suppressed). More than one –**e** option may be given. Only one *function-name* may be given with each –**e** option.

–**F** *function-name*
Print the graph profile entry only for routine *function-name* and its descendants (as –**f,** below) and also use only the times of the printed routines in total time and percentage computations. More than one –**F** option may be given. Only one *function-name* may be given with each –**F** option. The –**F** option overrides the –**E** option.

–**f** *function-name*
Print the graph profile entry only for routine *function-name* and its descendants. More than one –**f** option may be given. Only one *function-name* may be given with each –**f** option.

–n Limits the size of flat and graph profile listings to the top **n** offending functions.

–s Produce a profile file **gmon.sum** which represents the sum of the profile information in all of the specified profile files. This summary profile file may be given to subsequent executions of **gprof** (also with –**s**) to accumulate profile data across several runs of an **a.out** file. (See also the –**D** option.)

–z Display routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the –**c** option for discovering which routines were never called.

ENVIRONMENT

PROFDIR If this environment variable contains a value, place profiling output within that directory, in a file named *pid***.***programname*. *pid* is the process ID, and *programname* is the name of the program being profiled, as determined by removing any path prefix from the **argv[0]** with which the program was called. If the variable contains a NULL value, no profiling output is produced. Otherwise, profiling output is placed in the file **gmon.out**.

FILES

a.out	executable file containing namelist
gmon.out	dynamic call-graph and profile
gmon.sum	summarized dynamic call-graph and profile
$PROFDIR/*pid***.***programname*	

SEE ALSO

cc(1B), **prof**(1), **exit**(2), **profil**(2), **monitor**(3C), **prof**(5)

Graham, S.L., Kessler, P.B., McKusick, M.K., '*gprof: A Call Graph Execution Profiler*', *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call-graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

NAME

graph – draw a graph

SYNOPSIS

graph [**–a** *spacing* [*start*]] [**–b**] [**–c** *string*] [**–g** *gridstyle*] [**–l** *label*]
[**–m** *connectmode*] [**–s**] [**–x [l]** *lower* [*upper* [*spacing*]]]
[**–y** [**l**] *lower* [*upper* [*spacing*]]] [**–h** *fraction*] [**–w** *fraction*] [**–r** *fraction*]
[**–u** *fraction*] [**–t**] . . .

AVAILABILITY

SUNWesu

DESCRIPTION

graph with no options takes pairs of numbers from the standard input as abscissaes and ordinates of a graph. Successive points are connected by straight lines. The standard output from **graph** contains plotting instructions suitable for input to **plot**(1B) or to the command **lpr –g** (see **lpr**(1B)).

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "**. . .**", in which case they may be empty or contain blanks and numbers; labels never contain NEWLINE characters.

A legend indicating grid range is produced with a grid unless the **–s** option is present.

OPTIONS

Each option is recognized as a separate argument. If a specified lower limit exceeds the upper limit, the axis is reversed.

–a *spacing* **[** *start* **]**	Supply abscissaes automatically (they are missing from the input); *spacing* is the spacing (default 1). *start* is the starting point for automatic abscissaes (default 0 or lower limit given by **–x**).
–b	Break (disconnect) the graph after each label in the input.
–c *string*	*String* is the default label for each point.
–g *gridstyle*	*Gridstyle* is the grid style: 0 no grid, 1 frame with ticks, 2 full grid (default).
–l *label*	*label* is label for graph.
–m *connectmode*	Mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.
–s	Save screen, do not erase before plotting.
–x [l] *lower* **[** *upper* **[** *spacing* **]]**	If **l** is present, *x* axis is logarithmic. *lower* and *upper* are lower (and upper) *x* limits. *spacing*, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.
–y [l] *lower* **[** *upper* **[** *spacing* **]]**	If **l** is present, *y* axis is logarithmic. *lower* and *upper* are lower (and upper) *y* limits. *spacing*, if present, is grid spacing on *y* axis. Normally these quantities are determined automatically.

–**h** *fraction*	*fraction* of space for height.
–**w** *fraction*	*fraction* of space for width.
–**r** *fraction*	*fraction* of space to move right before plotting.
–**u** *fraction*	*fraction* of space to move up before plotting.
–**t**	Transpose horizontal and vertical axes. Option –**x** now applies to the vertical axis.

SEE ALSO

lpr(1B), **plot**(1B), **spline**(1)

BUGS

graph stores all points internally and drops those for which there is no room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

NAME

grep – search a file for a pattern

SYNOPSIS

grep [–**bchilnsvw**] *limited-regular-expression* [*filename* ...]

AVAILABILITY

SUNWcsu

DESCRIPTION

grep searches files for a pattern and prints all lines that contain that pattern. **grep** uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with **ed**(1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters **$**, ∗, **[**, **^**, | , **(**, **)**, and \ in the *limited-regular-expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited-regular-expression* in single quotes ′...′.

If no files are specified, **grep** assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

OPTIONS

–**b**	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
–**c**	Print only a count of the lines that contain the pattern.
–**h**	Prevents the name of the file containing the matching line from being appended to that line. Used when searching multiple files.
–**i**	Ignore upper/lower case distinction during comparisons.
–**l**	Print only the names of files with matching lines, separated by NEWLINE characters. Does not repeat the names of files when the pattern is found more than once.
–**n**	Precede each line by its line number in the file (first line is 1).
–**s**	Suppress error messages about nonexistent or unreadable files
–**v**	Print all lines except those that contain the pattern.
–**w**	Search for the expression as a word as if surrounded by \< and \>.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **grep** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **grep** behaves.

SEE ALSO

ed(1), **egrep**(1), **fgrep**(1), **sed**(1), **sh**(1), **environ**(5)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

NOTES

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**. If there is a line with embedded nulls, **grep** will only match up to the first null; if it matches, it will print the entire line.

NAME

groups – print group membership of user

SYNOPSIS

groups [*user* . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

The command **groups** prints on standard output the groups to which you or the optionally specified user belong. Each user belongs to a group specified in **/etc/passwd** and possibly to other groups as specified in **/etc/group**. Note that **/etc/passwd** specifies the numerical ID (**gid**) of the group. The **groups** command converts **gid** to the group name in the output.

EXAMPLE

The output takes the following form:

```
example% groups tester01 tester02
tester01 : staff
tester02 : staff
example%
```

FILES

/etc/passwd
/etc/group

SEE ALSO

group(4), **passwd**(4)

NAME

groups – display a user’s group memberships

SYNOPSIS

/usr/ucb/groups [*user* . . .]

AVAILABILITY

SUNWscpu

DESCRIPTION

With no arguments, **groups** displays the groups to which you belong; else it displays the groups to which the **user** belongs. Each user belongs to a group specified in the password file **/etc/passwd** and possibly to other groups as specified in the file **/etc/group**. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

FILES

/etc/passwd
/etc/group

SEE ALSO

getgroups(2)

NOTES

This command is obsolete.

NAME

grpck – check group database entries

SYNOPSIS

/usr/etc/grpck [*filename*]

DESCRIPTION

grpck checks that a file in **group**(4) does not contain any errors; it checks the **/etc/group** file by default.

FILES

/etc/group

SEE ALSO

groups(1), **group**(4), **passwd**(4)

DIAGNOSTICS

Too many/few fields
An entry in the group file does not have the proper number of fields.

No group name
The group name field of an entry is empty.

Bad character(s) in group name
The group name in an entry contains characters other than lower-case letters and digits.

Invalid GID
The group ID field in an entry is not numeric or is greater than 65535.

Null login name
A login name in the list of login names in an entry is null.

Logname not found in password file
A login name in the list of login names in an entry is not in the password file.

Line too long
A line (including the newline character) in the group file exceeds the maximum length of 512 characters.

Duplicate logname entry
A login name appears more than once in the list of login names for a group file entry.

Out of memory
The program cannot allocate memory in order to continue.

Maximum groups exceeded for logname
A login name's group membership exceeds the maximum, NGROUPS_MAX.

NAME

hash, rehash, unhash, hashstat – shell built-in functions to evaluate the internal hash table of the contents of directories

SYNOPSIS

sh

hash [**–r**] [*name* . . .]

csh

rehash

unhash

hashstat

ksh

hash

DESCRIPTION

sh

For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **–r** option to the **hash** built-in causes the shell to forget all remembered locations. If no arguments are given, **hash** provides information about remembered commands. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (∗) adjacent to the **Hits** information. *Cost* will be incremented when the recalculation is done.

csh

rehash recomputes the internal hash table of the contents of directories listed in the **path** environmental variable to account for new commands added.

unhash disables the internal hash table.

hashstat prints a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **exec**s). An **exec** is attempted for each component of the *path* where the hash function indicates a possible hit and in each component that does not begin with a ‘/’.

ksh

hash provides information about remembered commands. In the first column of the output, **Hits**, is the number of times a command has been invoked by the shell process. In the second column, **Cost**, is a measure of the work required to locate a command in the search path. The third column lists the command. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (∗) adjacent to the **Hits** information. **Cost** will be incremented when the recalculation is done.

SEE ALSO

csh(1), **ksh**(1), **sh**(1)

NAME

head – display first few lines of files

SYNOPSIS

head [–*n*] [*filename*. . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

head copies the first *n* lines of each *filename* to the standard output. If no *filename* is given, **head** copies lines from the standard input. The default value of *n* is 10 lines.

When more than one file is specified, the start of each file will look like:

```
==> filename <==
```

Thus, a common way to display a set of short files, identifying each one, is:

```
example% head –9999 filename1 filename2 . . .
```

SEE ALSO

cat(1), **more**(1), **pg**(1), **tail**(1)

NAME

history, fc – shell built-in functions to re-use previous command-lines from the current shell

SYNOPSIS

csh

history [–**hr**] [*n*]

ksh

fc [–**e** *ename*] –**nlr**] [*first* [*last*]]
fc –**e** – [*old*=*new*] [*command*]

DESCRIPTION

csh

Display the history list; if *n* is given, display only the *n* most recent events.

–**r**	Reverse the order of printout to be most recent first rather than oldest first.
–**h**	Display the history list without leading numbers. This is used to produce files suitable for sourcing using the –**h** option to the **csh** built-in command, **source**(1).

ksh

fc, in the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the –**l** flag is selected, the commands are listed on standard output. Otherwise, the editor program –**e** *name* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the variable **FCEDIT** (default **/bin/ed**) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and –16 for listing. The flag –**r** reverses the order of the commands and the flag –**n** suppresses command numbers when listing. In the second form the *command* is re-executed after the substitution *old*=*new* is performed.

SEE ALSO

csh(1), **ksh**(1), **sh**(1), **source**(1)

NAME

hostid – print the numeric identifier of the current host

SYNOPSIS

/usr/ucb/hostid

AVAILABILITY

SUNWscpu

DESCRIPTION

The **hostid** command prints the identifier of the current host in hexadecimal. This numeric value is likely to differ when **hostid** is run on a different machine.

SEE ALSO

sysinfo(2), **gethostid**(3B)

NAME

hostname – set or print name of current host system

SYNOPSIS

/usr/ucb/hostname [*name-of-host*]

AVAILABILITY

SUNWscpu

DESCRIPTION

The **hostname** command prints the name of the current host, as given before the **login** prompt. The super-user can set the hostname by giving an argument.

SEE ALSO

uname(1)

NAME

iconv – code set conversion utility

SYNOPSIS

iconv **–f** *fromcode* **–t** *tocode* [*filename*]

AVAILABILITY

SUNWcsu

DESCRIPTION

iconv converts the characters or sequences of characters in *filename* from one code set to another and writes the results to standard output. Should no conversion exist for a particular character then it is converted to the underscore '_' in the target codeset.

The required arguments *fromcode* and *tocode* identify the input and output code sets, respectively. If no *filename* argument is specified on the command line, **iconv** reads the standard input.

iconv will always convert to or from the ISO 8859-1 Latin alphabet No.1, from or to an ISO 646 ASCII variant codeset for a particular language. The ISO 8859-1 codeset will support the majority of 8 bit codesets. The conversions attempted by **iconv** accommodate the most commonly used languages.

The following table lists the supported conversions.

Code Set Conversions Supported				
Code	Symbol	Target Code	Symbol	comment
ISO 646	646	ISO 8859-1	8859	US ASCII
ISO 646de	646de	ISO 8859-1	8859	German
ISO 646da	646da	ISO 8859-1	8859	Danish
ISO 646en	646en	ISO 8859-1	8859	English ASCII
ISO 646es	646es	ISO 8859-1	8859	Spanish
ISO 646fr	646fr	ISO 8859-1	8859	French
ISO 646it	646it	ISO 8859-1	8859	Italian
ISO 646sv	646sv	ISO 8859-1	8859	Swedish
ISO 8859-1	8859	ISO 646	646	7 bit ASCII
ISO 8859-1	8859	ISO 646de	646de	German
ISO 8859-1	8859	ISO 646da	646da	Danish
ISO 8859-1	8859	ISO 646en	646en	English ASCII
ISO 8859-1	8859	ISO 646es	646es	Spanish
ISO 8859-1	8859	ISO 646fr	646fr	French
ISO 8859-1	8859	ISO 646it	646it	Italian
ISO 8859-1	8859	ISO 646sv	646sv	Swedish

The conversions are performed according to the tables found on **iconv**(5).

OPTIONS

–f *fromcode*	Identifies the input code set.
–t *tocode*	Identifies the output code set.

EXAMPLES

The following converts the contents of file **mail1** from code set **8859** to **646fr** and stores the results in file *mail.local*.

```
example% iconv –f 8859 –t 646fr mail1 > mail.local
```

FILES

/usr/lib/iconv/iconv_data	lists the conversions supported
/usr/lib/iconv/*.t	conversion tables

SEE ALSO

iconv(5)

DIAGNOSTICS

iconv returns 0 upon successful completion, 1 otherwise.

NAME

if, test – shell built-in functions to evaluate expression(s) or to make execution of actions dependent upon the evaluation of expression(s)

SYNOPSIS

sh

if *condition*; **then** *action* ; **[elif** *condition* ; **then** *action* **] . . . [else** *action* **]** ; **fi**

test *expression*
[*expression* **]**

csh

if (*expr***)** *command*

if (*expr***) then**
        commands . . .
else if (*expr2***) then**
        commands . . .
else
        commands . . .
endif

ksh

if *condition* ; **then** *action* [[; **elif** *condition* ; **then** *action*] . . .] ; **else** *action* ; **fi**

test *expression*
[*expression* **]**

DESCRIPTION

sh

The *condition* following **if** is executed and, if it returns a zero exit status, the *action* following the first **then** is executed. Otherwise, the *condition* following **elif** is executed and, if its value is zero, the *action* following the next **then** is executed. Failing that, the **else** *action* is executed. If no **else** *action* or **then** *action* is executed, then the **if** command returns a zero exit status.

test evaluates the expression *expression* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; **test** also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second **SYNOPSIS** line) must be separate arguments to the **test** command; normally these items are separated by spaces.

Primitives:
The following primitives are used to construct *expression*:

–**r** *filename*	True if *filename* exists and is readable.
–**w** *filename*	True if *filename* exists and is writable.
–**x** *filename*	True if *filename* exists and is executable.
–**f** *filename*	True if *filename* exists and is a regular file. Alternatively, if **/usr/bin/sh** users specify **/usr/ucb** before **/usr/bin** in their PATH environment

variable, then **test** will return true if *filename* exists and is (**not**–**a**–**directory**). This is also the default for **/usr/bin/csh** users.

–**d** *filename*	True if *filename* exists and is a directory.
–**h** *filename*	True if *filename* exists and is a symbolic link. With all other primitives (except –**L** *filename*), the symbolic links are followed by default.
–**c** *filename*	True if *filename* exists and is a character special file.
–**b** *filename*	True if *filename* exists and is a block special file.
–**p** *filename*	True if *filename* exists and is a named pipe (fifo).
–**u** *filename*	True if *filename* exists and its set-user-ID bit is set.
–**g** *filename*	True if *filename* exists and its set-group-ID bit is set.
–**k** *filename*	True if *filename* exists and its sticky bit is set.
–**s** *filename*	True if *filename* exists and has a size greater than zero.
–**t** [*fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–**z** *s1*	True if the length of string *s1* is zero.
–**n** *s1*	True if the length of the string *s1* is non-zero.
s1 **=** *s2*	True if strings *s1* and *s2* are identical.
s1 **!=** *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.
n1 –**eq** *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons –**ne**, –**gt**, –**ge**, –**lt**, and –**le** may be used in place of –**eq**.
–**L** *filename*	True if *filename* exists and is a symbolic link. With all other primitives (except –**h** *filename*), the symbolic links are followed by default.

Operators:
These primaries may be combined with the following operators:

!	Unary negation operator.
–**a**	Binary *and* operator.
–**o**	Binary *or* operator (–**a** has higher precedence than –**o**).
(*expression*)	Parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

The **not**–**a**–**directory** alternative to the –**f** option is a transition aid for BSD applications and may not be supported in future releases.

The –**L** option is a migration aid for users of other shells which have similar options and may not be supported in future releases.

If you test a file you own (the –**r** –**w** or –**x** tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the **group** or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The **=** and **!=** operators have a higher precedence than the –**r** through –**n** operators, and **=** and **!=** always expect arguments; therefore, **=** and **!=** cannot be used with the –**r** through –**n** operators.

If more than one argument follows the –**r** through –**n** operators, only the first argument is examined; the others are ignored, unless a –**a** or a –**o** is the second argument.

csh

With the one-line form of **if**, if the specified expression evaluates to true, the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Note: I/O redirection occurs even if *expr* is false, when *command* is *not* executed (this is a bug).

With the multi-line form of **if**, if *expr* is true, *commands* up to the first **else** are executed. Otherwise, if *expr2* is true, the *commands* between the **else if** and the second **else** are executed. Otherwise, *commands* between the **else** and the **endif** are executed. Any number of **else if** pairs are allowed, but only one **else**. Only one **endif** is needed, but it is required. The words **else** and **endif** must be the first nonwhite characters on a line. The **if** must appear alone on its input line or after an **else**.

ksh

The *condition* following **if** is executed and, if it returns an exit status of zero, the *action* following the first **then** is executed. Otherwise, the *condition* following **elif** is executed and, if its value is zero, the *action* following the next **then** is executed. Failing that, the **else** *action* is executed. If no **else** *action* or **then** *action* is executed, then the **if** command returns a zero exit status.

For a description of the **test** built-in, see the (**sh**) Bourne shell's **test** built-in above.

EXAMPLES

In the **if** command examples, three conditions are tested, and if all three evaluate as true or successful, then their validities are printed to the screen. The two forms of the **test** built-in follow the Bourne shell's **if** example.

sh

```
ZERO=0 ONE=1 TWO=2 ROOT=root
if  [ $ONE -gt $ZERO ]  &&
[ $TWO -eq 2 ] &&
[ ‘grep $ROOT  /etc/passwd >&1 /dev/null‘ ]
then
     echo "$ONE is greater than zero, $TWO equals 2, and root is a user-name
             in the password file"
else
     echo "At least one of the three test conditions is false"
fi
```

Examples of the **test** built-in:

```
test ‘grep $ROOT /etc/passwd >&1 /dev/null‘
echo $?    /* test for success */

[ ‘grep nosuchname /etc/passwd >&1 /dev/null‘ ]
echo $?    /* test for failure */
```

csh

```
@ ZERO = 0; @ ONE = 1; @ TWO = 2;  set ROOT = root
grep $ROOT  /etc/passwd >&1 /dev/null
if ( "$status" == "0" && $ONE > $ZERO && $TWO == 2 ) then
     /* $status must be tested for immediately following grep */
     echo "$ONE is greater than zero, $TWO equals 2, and root is a user-name
             in the password file"
endif
```

ksh

```
ZERO=0 ONE=1 TWO=2 ROOT=root
if  [ $ONE -gt $ZERO ] ; [ $TWO -eq 2 ] ;
[ ‘grep $ROOT  /etc/passwd >&1 /dev/null‘ ] ;
then
     echo "$ONE is greater than zero, $TWO equals 2, and root is a user-name
             in the password file"
fi
```

The Korn shell will also accept the syntax of both the **if** command and the **test** command of the Bourne shell.

When using the brackets (**[ ]**) within **if** commands, you must separate both ends of both brackets from other characters with a space.

SEE ALSO

csh(1), **ksh**(1), **sh**(1), **test**(1B)

NOTES

Both the Bourne shell, **sh**, and the Korn shell, **ksh**, can use the semicolon and the carriage return interchangeably in their syntax of the **if**, **for**, and **while** built-in commands.

NAME

indicator – display application specific alarms and / or the "working" indicator

SYNOPSIS

indicator [–**b** [*n*]] [–**c** *column*] [–**l** *length*] [–**o**] [–**w**] [*string* . . .]

DESCRIPTION

The **indicator** function displays application specific alarms or the "working" indicator, or both, on the FMLI banner line. By default, **indicator** ???? The argument *string* is a string to be displayed on the banner line, and should always be the last argument given. Note that *string* is not automatically cleared from the banner line.

OPTIONS

–**b***n*	The –**b** option rings the terminal bell *n* times, where *n* is an integer from 1 to 10. The default value is 1. If the terminal has no bell, the screen is flashed instead, if possible.
–**c** *column*	The –**c** option defines the column of the banner line at which to start the indicator string. The argument *column* must be an integer from **0** to **DISPLAYW-1 .** If the –**c** option is not used, *column* defaults to **0 .**
–**l** *length*	The –**l** option defines the maximum length of the string displayed. If *string* is longer than *length* characters, it will be truncated. The argument *length* must be an integer from **1** to **DISPLAYW .** If the –**l** option is not used, *length* defaults to **DISPLAYW .** NOTE: if *string* doesn’t fit it will be truncated.
–**o**	The –**o** option causes **indicator** to duplicate its output to *stdout* .
–**w**	The –**w** option turns on the "working" indicator.

EXAMPLES

When the value entered in a form field is invalid, the following use of **indicator** will ring the bell three times and display the word WRONG starting at column 1 of the banner line.

```
invalidmsg=`indicator –b 3 –c 1 "WRONG"`
```

To clear the indicator after telling the user the entry is wrong:

```
invalidmsg=`indicator –b 9 –c 1 "WRONG"; sleep 3;
        indicator –c 1 "     "`
```

In this example the value of **invalidmsg** (in this case the default value **Input is not valid**), still appears on the FMLI message line.

NAME

indxbib – create an inverted index to a bibliographic database

SYNOPSIS

indxbib *database-file*. . .

AVAILABILITY

SUNWdoc

DESCRIPTION

indxbib makes an inverted index to the named *database-file* (which must reside within the current directory), typically for use by **lookbib**(1) and **refer**(1). A *database* contains bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '**%**', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '**%**'.

indxbib is a shell script that calls two programs: **/usr/lib/refer/mkey** and **/usr/lib/refer/inv**. **mkey** truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1000 or > 2099. These parameters can be changed.

indxbib creates an entry file (with a **.ia** suffix), a posting file (**.ib**), and a tag file (**.ic**), in the working directory.

FILES

/usr/lib/refer/mkey
/usr/lib/refer/inv

*x***.ia**	entry file
*x***.ib**	posting file
*x***.ic**	tag file
*x***.ig**	reference file

SEE ALSO

addbib(1), **lookbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1)

BUGS

All dates should probably be indexed, since many disciplines refer to literature written in the 1800s or earlier.

indxbib does not recognize pathnames.

NAME

install – install files

SYNOPSIS

/usr/ucb/install [**–cs**] [**–g** *group*] [**–m** *mode*] [**–o** *owner*] *filename1 filename2*

/usr/ucb/install [**–cs**] [**–g** *group*] [**–m** *mode*] [**–o** *owner*] *filename . . . directory*

/usr/ucb/install –d [**–g** *group*] [**–m** *mode*] [**–o** *owner*] *directory*

AVAILABILITY

SUNWscpu

DESCRIPTION

Install is used within makefiles to copy new versions of files into a destination directory and to create the destination directory itself.

The first two forms are similar to the **cp**(1) command with the addition that executable files can be stripped during the copy and the owner, group, and mode of the installed file(s) can be given.

The third form can be used to create a destination directory with the required owner, group and permissions.

Note: **install** uses no special privileges to copy files from one place to another. The implications of this are:

- You must have permission to read the files to be installed.
- You must have permission to copy into the destination file or directory.
- You must have permission to change the modes on the final copy of the file if you want to use the **–m** option to change modes.
- You must be superuser if you want to specify the ownership of the installed file with **–o**. If you are not the super-user, or if **–o** is not in effect, the installed file will be owned by you, regardless of who owns the original.

OPTIONS

–c	Copy files. In fact **install** *always* copies files, but the **–c** option is retained for backwards compatibility with old shell scripts that might otherwise break.
–d	Create a directory. Missing parent directories are created as required as in **mkdir –p**. If the directory already exists, the owner, group and mode will be set to the values given on the command line.
–s	Strip executable files as they are copied.
–g *group*	Set the group ownership of the installed file or directory. (staff by default.)
–m *mode*	Set the mode for the installed file or directory. (0755 by default.)
–o *owner*	If run as root, set the ownership of the installed file to the user-ID of *owner*.

SEE ALSO

chgrp(1), **chmod**(1), **chown**(1), **cp**(1), **mkdir**(1), **strip**(1), **install**(1M)

NAME

ipcrm – remove a message queue, semaphore set, or shared memory ID

SYNOPSIS

ipcrm [–**m** *shmid*] [–**q** *msqid*] [–**s** *semid*] [–**M** *shmkey*] [–**Q** *msgkey*] [–**S** *semkey*]

AVAILABILITY

SUNWipc

DESCRIPTION

ipcrm removes one or more messages, semaphores, or shared memory identifiers.

OPTIONS

The identifiers are specified by the following *options* :

–**m** *shmid*	Remove the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
–**q** *msqid*	Remove the message queue identifier *msqid* from the system and destroy the message queue and data structure associated with it.
–**s** *semid*	Remove the semaphore identifier *semid* from the system and destroy the set of semaphores and data structure associated with it.
–**M** *shmkey*	Removes the shared memory identifier, created with key *shmkey* , from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
–**Q** *msgkey*	Remove the message queue identifier, created with key *msgkey* , from the system and destroy the message queue and data structure associated with it.
–**S** *semkey*	Remove the semaphore identifier, created with key *semkey* , from the system and destroy the set of semaphores and data structure associated with it.

The details of the removes are described in **msgctl**(2), **shmctl**(2), and **semctl**(2). Use the **ipcs** command to find the identifiers and keys.

SEE ALSO

ipcs(1), **msgctl**(2), **msgget**(2), **msgop**(2), **semctl**(2), **semget**(2), **semop**(2), **shmctl**(2), **shmget**(2), **shmop**(2)

NAME

ipcs – report inter-process communication facilities status

SYNOPSIS

ipcs [−**abcmopqst**] [−**C** *corefile*] [−**N** *namelist*]

AVAILABILITY

SUNWipc

DESCRIPTION

ipcs prints information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system.

The information that is displayed is controlled by the options supplied.

OPTIONS

−**m**	Print information about active shared memory segments.
−**q**	Print information about active message queues.
−**s**	Print information about active semaphores.

If −**q**, −**m**, or −**s** are specified, information about only those indicated is printed. If none of these three are specified, information about all three is printed subject to these options:

−**a**	Use all print options. (This is a shorthand notation for −**b**, −**c**, −**o**, −**p**, and −**t**.)
−**b**	Print information on biggest allowable size: maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores. See below for meaning of columns in a listing.
−**c**	Print creator's login name and group name. See below.
−**o**	Print information on outstanding usage: number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.
−**p**	Print process number information: process ID of last process to send a message, process ID of last process to receive a message on message queues, process ID of creating process, and process ID of last process to attach or detach on shared memory segments. See below.
−**t**	Print time information: time of the last control operation that changed the access permissions for all facilities, time of last **msgsnd** and last **msgrcv** on message queues, time of last **shmat** and last **shmdt** on shared memory, time of last **semop** on semaphores. See below.
−**C** *corefile*	Use the file *corefile* in place of **/dev/mem** and **/dev/kmem**. Use a core dump obtained from **savecore(1M)** in place of **/dev/mem** and **/dev/kmem**. Without the −**C** option (default), the running system image is used.
−**N** *namelist*	Use the file *namelist* in place of **/dev/ksyms**.

The column headings and the meaning of the columns in an **ipcs** listing are given below; the letters in parentheses indicate the options that cause the corresponding heading to appear; ‘‘all’’ means that the heading always appears. Note: These options only determine what information is provided for each facility; they do not determine which facilities are listed.

T	(all)	Type of the facility: **q** message queue **m** shared memory segment **s** semaphore
ID	(all)	The identifier for the facility entry.
KEY	(all)	The key used as an argument to **msgget**, **semget**, or **shmget** to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)
MODE	(all)	The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows. The first two characters are: **R** A process is waiting on a *msgrcv*. **S** A process is waiting on a *msgsnd*. **D** The associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it. **C** The associated shared memory segment is to be cleared when the first attach is executed. – The corresponding special flag is not set. The next nine characters are interpreted as three sets of three bits each. The first set refers to the owner’s permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused. The permissions are indicated as follows: **r** Read permission is granted. **w** Write permission is granted. **a** Alter permission is granted. – The indicated permission is not granted.
OWNER	(all)	The login name of the owner of the facility entry.
GROUP	(all)	The group name of the group of the owner of the facility entry.
CREATOR	(a,c)	The login name of the creator of the facility entry.
CGROUP	(a,c)	The group name of the group of the creator of the facility entry.
CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	The number of messages currently outstanding on the associated

		message queue.
QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p)	The process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	The time the last message was sent to the associated queue.
RTIME	(a,t)	The time the last message was received from the associated queue.
CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/etc/group	group names
/etc/passwd	user names
/dev/mem	memory
/dev/ksyms	system namelist

SEE ALSO

msgop(2), **semop**(2), **shmop**(2)

NOTES

If the user specifies either the –**C** or –**N** flag, the real and effective UID / GID is set to the real UID / GID of the user invoking **ipcs**.

Things can change while **ipcs** is running; the information it gives is guaranteed to be accurate only when it was retrieved.

NAME

jobs, fg, bg, stop, notify – shell built-in functions to control process execution

SYNOPSIS

sh

jobs [**–p | –l**] [**%***jobid* ...]

jobs –x *command* [*arguments*]

fg [**%***jobid* . . .]

bg [**%***jobid* . . .]

stop %*jobid* . . .

stop *pid* . . .

csh

jobs.[**–l**]

fg [**%***job*]

bg [**%***job*] . . .

notify [**%***job*] . . .

stop %*jobid* . . .

stop *pid* . . .

ksh

jobs [**–lnp**] [**%***job* . . .]

fg [**%***job* . . .]

bg [**%***job* . . .]

stop %*jobid* . . .

stop *pid* . . .

DESCRIPTION

sh

When Job Control is enabled, **jobs** reports all jobs that are stopped or executing in the background. If **%***jobid* is omitted, all jobs that are stopped or running in the background will be reported. The following options will modify / enhance the output of **jobs**:

–l	Report the process group ID and working directory of the jobs.
–p	Report only the process group ID of the jobs.
–x	Replace any *jobid* found in *command* or *arguments* with the corresponding process group ID, and then execute *command* passing it *arguments*.

When the shell is invoked as **jsh**, Job Control is enabled in addition to all of the functionality described previously for **sh**. Typically Job Control is enabled for the interactive shell only. Non-interactive shells typically do not benefit from the added functionality of Job Control.

With Job Control enabled every command or pipeline the user enters at the terminal is called a *job*. All jobs exist in one of the following states: foreground, background or stopped. These terms are defined as follows: 1) a job in the foreground has read and write access to the controlling terminal; 2) a job in the background is denied read access and has conditional write access to the controlling terminal (see **stty**(1)); 3) a stopped job is a job that has been placed in a suspended state, usually as a result of a **SIGTSTP** signal (see **signal**(5)).

Every job that the shell starts is assigned a positive integer, called a *job number* which is tracked by the shell and will be used as an identifier to indicate a specific job. Additionally the shell keeps track of the *current* and *previous* jobs. The *current job* is the most recent job to be started or restarted. The *previous job* is the first non-current job.

The acceptable syntax for a Job Identifier is of the form:

%*jobid*

where, *jobid* may be specified in any of the following formats:

% or **+**	for the current job
–	for the previous job
?<*string*>	specify the job for which the command line uniquely contains *string*.
n	for job number *n*, where *n* is a job number
pref	where *pref* is a unique prefix of the command name (for example, if the command **ls –l name** were running in the background, it could be referred to as **%ls**); *pref* cannot contain blanks unless it is quoted.

When Job Control is enabled, **fg** resumes the execution of a stopped job in the foreground, also moves an executing background job into the foreground. If **%***jobid* is omitted the current job is assumed.

When Job Control is enabled, **bg** resumes the execution of a stopped job in the background. If **%***jobid* is omitted the current job is assumed.

stop stops the execution of a background job(s) by using its *jobid*, or of any process by using its *pid*. (see **ps**(1))

csh

jobs, without an argument, lists the active jobs under job control.

–l List process IDs, in addition to the normal information.

The shell associates a numbered *job* with each command sequence to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically CTRL-z). When a command or command sequence (semicolon separated list) is started in the background using the **&** metacharacter, the shell displays a line with the job number in brackets and a list of associated process numbers:

[1] 1234

To see the current list of jobs, use the **jobs** built-in command. The job most recently stopped (or put into the background if none are stopped) is referred to as the *current* job and is indicated with a '**+**'. The previous job is indicated with a '–'; when the current job is terminated or moved to the foreground, this job takes its place (becomes the new current job).

To manipulate jobs, refer to the **bg**, **fg**, **kill**, **stop**, and **%** built-in commands.

A reference to a job begins with a '**%**'. By itself, the percent-sign refers to the current job.

% %+ %%	The current job.
%–	The previous job.
%*j*	Refer to job *j* as in: '**kill –9 %***j*'. *j* can be a job number, or a string that uniquely specifies the command line by which it was started; '**fg %vi**' might bring a stopped **vi** job to the foreground, for instance.
%?*string*	Specify the job for which the command line uniquely contains *string*.

A job running in the background stops when it attempts to read from the terminal. Background jobs can normally produce output, but this can be suppressed using the '**stty tostop**' command.

fg brings the current or specified *job* into the foreground.

bg runs the current or specified jobs in the background.

stop stops the execution of a background job(s) by using its *jobid*, or of any process by using its *pid*. (see **ps**(1)).

notify will notify the user asynchronously when the status of the current job or specified jobs changes.

ksh

jobs lists information about each given job; or all active jobs if *job* is omitted. The **–l** flag lists process ids in addition to the normal information. The **–n** flag only displays jobs that have stopped or exited since last notified. The **–p** flag causes only the process group to be listed. If the **monitor** option of the **set** command is turned on, an interactive shell associates a **job** with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the **job**, which was started asynchronously, was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key **^Z** (CTRL-Z) which sends a **STOP** signal to the current job. The shell will then normally indicate that the job has been **'Stopped'**, and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A **^Z** takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command **"stty tostop"**. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to **job**s in the shell. A **job** can be referred to by the process id of any process of the **job** or by one of the following:

%*number*	The job with the given number.
%*string*	Any job whose command line begins with *string*.
%?*string*	Any job whose command line begins with *string*.
%?*string*	Any job whose command line contains *string*.
%%	Current job.
%+	Equivalent to **%%**.
%–	Previous job.

The shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. When the monitor mode is on, each background job that completes triggers any trap set for **CHLD**. When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped(running) jobs.' You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

The **fg** command is only on systems that support job control. Each *job* specified is brought to the foreground. Otherwise, the current job is brought into the foreground.

The **bg** command is only on systems that support job control. **bg** puts each specified *job* into the background. The current job is put in the background if *job* is not specified.

stop stops the execution of a background job(s) by using its *jobid*, or of any process by using its *pid*. (see **ps**(1))

SEE ALSO

csh(1), **kill**(1), **ksh**(1), **ps**(1), **sh**(1), **stop**(1), **shell_builtins**(1), **stty**(1) **signal**(5)

NAME

join – relational database operator

SYNOPSIS

join [**–a***n*] [**–e** *s*] [**–j***n m*] [**–o** *list*] [**–t***c*] *filename1 filename2*

AVAILABILITY

SUNWcsu

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *filename1* and *filename2*. If *filename1* is **–** , the standard input is used.

filename1 and *filename2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line (see **sort**(1)).

There is one line in the output for each pair of lines in *filename1* and *filename2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *filename1*, then the rest of the line from *filename2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

OPTIONS

Some of the options below use the argument *n*. This argument should be a **1** or a **2** referring to either *filename1* or *filename2*, respectively.

–a*n*	In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
–e *s*	Replace empty output fields with string *s*.
–j*n m*	Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.
–o *list*	Each output line includes the fields specified in *list*, each element of which has the form *n***.***m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
–t*c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLES

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
example% join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group
```

ENVIRONMENT

If any of the **LC_∗** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **join** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_∗** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **join** behaves.

LC_CTYPE

Determines how **join** handles characters. When **LC_CTYPE** is set to a valid value, **join** can display and handle text and filenames containing valid characters for that locale. **join** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **join** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

awk(1), **comm**(1), **sort**(1), **uniq**(1), **environ**(5)

NOTES

With default field separation, the collating sequence is that of **sort –b**; with **–t**, the sequence is that of a plain sort.

The conventions of the **join**, **sort**, **comm**, **uniq**, and **awk** commands are wildly incongruous.

Filenames that are numeric may cause conflict when the **–o** option is used just before listing filenames.

NAME

kbd – manipulate the state of keyboard or display the type of keyboard

SYNOPSIS

kbd [**–r**] [**–t**] [**–c on** | **off**] [**–d** *keyboard device*]

AVAILABILITY

SUNWcsu

DESCRIPTION

kbd manipulates the state of the keyboard, or displays the keyboard type. The default keyboard device being set is **/dev/kbd**.

Only keyboards that support a clicker respond to the **–c** option. If you want to turn clicking on by default, this can be set in the **/etc/rc**∗ files.

OPTIONS

–r Reset the keyboard as if power-up.

–t Return the type of the keyboard being used.

–c *on/off state*
Turn the clicking of the keyboard on or off.

on	Enable clicking.
off	Disable clicking.

–d *keyboard device*
Specify the keyboard device being set. The default is **/dev/kbd**.

EXAMPLES

The following example displays the keyboard type.

```
example% kbd –t
type 4 Sun keyboard
```

To enable clicking by default, add the following line to your **/etc/rc**∗ file.

```
kbd –c on
```

FILES

/etc/rc∗
/dev/kbd

SEE ALSO

loadkeys(1), **keytables**(4), **kb**(7)

BUGS

There is no way to determine the state of the keyboard click setting.

NAME

kdestroy – destroy Kerberos tickets

SYNOPSIS

/usr/bin/kdestroy [**–fnq**]

AVAILABILITY

SUNWcsu

DESCRIPTION

kdestroy destroys the user's active Kerberos authorization tickets by writing zeros to the file that contains them. If the ticket file does not exist, **kdestroy** displays a message to that effect.

After overwriting the file, **kdestroy** removes the file from the system. The utility displays a message indicating the success or failure of the operation. If **kdestroy** is unable to destroy the ticket file, it will warn you by making your terminal beep.

In addition to removing the ticket file, **kdestroy** also invalidates all Kerberos credentials for this user being held in the kernel for use with NFS requests.

If desired, you can place the **kdestroy** command in your **.logout** file so that your tickets are destroyed automatically when you logout. Note, however, that doing this will cause NFS operations done on your behalf to fail after you logout.

OPTIONS

–f	Do not display the status message.
–n	Do not invalidate NFS credentials in the kernel. The credentials will continue to be valid until their normal expiration time, although new ones cannot be obtained until **kinit**(1) is run again for this user.
–q	Do not make your terminal beep if **kdestroy** fails to destroy the tickets.

FILES

The file specified by the **KRBTKFILE** environment variable if set, otherwise **/tmp/tkt***uid*

SEE ALSO

kerberos(1), **kinit**(1), **klist**(1)

BUGS

Only the tickets in the user's current ticket file are destroyed. Separate ticket files are used to hold root instance and password changing tickets. These files should probably be destroyed too, or all of a user's tickets should be kept in a single ticket file.

AUTHORS

Steve Miller, MIT Project Athena / Digital Equipment Corporation
Clifford Neuman, MIT Project Athena
Bill Sommerfeld, MIT Project Athena

NAME

kerberos – introduction to the Kerberos system

DESCRIPTION

The Kerberos system authenticates individual users in a network environment. After authenticating yourself to Kerberos, you can use the **kerberos** authentication option of network services such as NFS. In addition, in some environments you can use network utilities such as **rlogin**(1), **rcp**(1), and **rsh**(1) without having to present passwords to remote hosts and without having to bother with **.rhosts** files. See your system administrator for more information about Kerberos support at your site.

Before you can use Kerberos, you must be registered as a user in the Kerberos database. You can use the **kinit**(1) command to find out your status. This command tries to log you into the Kerberos system. **kinit** will prompt you for a username and password. Enter your username and password. If the utility lets you login without giving you a message, you have already been registered.

If you enter your username and **kinit** responds with this message:

Principal unknown (kerberos)

you haven't been registered as a Kerberos user. See your system administrator.

A Kerberos name contains three parts. The first is the *principal name*, which is usually a user's or service's name. The second is the *instance*, which in the case of a user is usually NULL. Some users may have privileged instances, however, such as **root** or **admin**. In the case of a service, the instance is the name of the machine on which it runs; that is, there can be an NFS service running on the machine ABC, which is different from the NFS service running on the machine XYZ. The third part of a Kerberos name is the *realm*. The realm corresponds to the Kerberos service providing authentication for the principal. For example, at MIT there is a Kerberos running at the Laboratory for Computer Science and one running at Project Athena.

When writing a Kerberos name, the principal name is separated from the instance (if not NULL) by a period, and the realm (if not the local realm) follows, preceded by an ‘‘@’’ sign. The following are examples of valid Kerberos names:

billb
jis.admin
srz@lcs.mit.edu
treese.root@athena.mit.edu

When you authenticate yourself with Kerberos, typically through the **kinit** command, Kerberos gives you an initial Kerberos *ticket*. (A Kerberos ticket is an encrypted protocol message that provides authentication.) Kerberos uses this ticket for network utilities such as NFS, **rlogin** and **rcp**. The ticket transactions are done transparently, so you do not have to worry about their management.

Note, however, that tickets expire. Privileged tickets, such as root instance tickets, expire in a few minutes, while tickets that carry more ordinary privileges may be good for several hours or a day, depending on the installation's policy. If your login session extends beyond the time limit, you will have to re-authenticate yourself to Kerberos to get new tickets. Use the **kinit** command to re-authenticate yourself.

If you use the **kinit** command to get your tickets, you can use the **kdestroy**(1) command to destroy your tickets before you end your login session. For more information about the **kinit** and **kdestroy** commands, see the **kinit**(1) and **kdestroy**(1) manual pages.

Currently, Kerberos supports NFS and other RPC network services using the **AUTH_KERB** authentication type. In some environments, the following network services are also supported: **rlogin**, **rsh**, and **rcp**. Other services are being worked on, such as the **pop** mail system, but are not yet available.

SEE ALSO

kdestroy(1), **kinit**(1), **klist**(1), **kerbd**(1M), **kerberos**(3N), **krb.conf**(4)

BUGS

Kerberos will not do authentication forwarding. In other words, if you use **rlogin** to login to a remote host, you cannot use Kerberos services from that host until you authenticate yourself explicitly on that host. Although you may need to authenticate yourself on the remote host, be aware that when you do so, **rlogin** sends your password across the network in clear text.

AUTHORS

Steve Miller, MIT Project Athena / Digital Equipment Corporation
Clifford Neuman, MIT Project Athena

The following people helped out on various aspects of the system:

Jeff Schiller designed and wrote the administration server and its user interface, **kadmin**. He also wrote the **dbm** version of the database management system.

Mark Colan developed the Kerberos versions of **rlogin**, **rsh**, and **rcp**, as well as contributing work on the servers.

John Ostlund developed the Kerberos versions of **passwd** and **userreg**.

Stan Zanarotti pioneered Kerberos in a foreign realm (LCS), and made many contributions based on that experience.

Many people contributed code and / or useful ideas. These include, Jim Aspnes, Bob Baldwin, John Barba, Richard Basch, Jim Bloom, Bill Bryant, Rob French, Dan Geer, David Jedlinsky, John Kohl, John Kubiatowicz, Bob McKie, Brian Murphy, Ken Raeburn, Chris Reed, Jon Rochlis, Mike Shanzer, Bill Sommerfeld, Jennifer Steiner, Ted Ts'o, and Win Treese.

RESTRICTIONS

COPYRIGHT 1985,1986 Massachusetts Institute of Technology

NAME

keylogin – decrypt and store secret key with keyserv

SYNOPSIS

/usr/bin/keylogin [**–r**]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **keylogin** command prompts for a password, and uses it to decrypt the user's secret key. The key may be found in the **/etc/publickey** file (see **publickey**(4)) or the NIS map ''publickey.byname'' or the NIS+ table ''cred.org_dir'' in the user's home domain. The sources and their lookup order are specified in the **/etc/nsswitch.conf** file (see **nsswitch.conf**(4)). Once decrypted, the user's secret key is stored by the local key server process, **keyserv**(1M). This stored key is used when issuing requests to any secure RPC services, such as NFS or NIS+. The program **keylogout**(1) can be used to delete the key stored by **keyserv.**

keylogin will fail if it cannot get the caller's key, or the password given is incorrect. For a new user or host, a new key can be added using **newkey**(1M), **nisaddcred**(1M), or **nisclient**(1M).

OPTIONS

–r Update the **/etc/.rootkey** file. This file holds the unencrypted secret key of the super-user. Only the super-user may use this option. It is used so that processes running as super-user can issue authenticated requests without requiring that the administrator explicitly run **keylogin** as super-user at system startup time (see **keyserv**(1M)). The **–r** option should be used by the administrator when the host's entry in the publickey database has changed, and the **/etc/.rootkey** file has become out-of-date with respect to the actual key pair stored in the publickey database. The permissions on the **/etc/.rootkey** file are such that it may be read and written by the super-user but by no other user on the system.

FILES

/etc/.rootkey	super-user's secret key

SEE ALSO

chkey(1), **keylogout**(1), **login**(1), **keyserv**(1M), **newkey**(1M), **nisaddcred**(1M), **nisclient**(1M), **publickey**(4), **nsswitch.conf**(4)

NAME

keylogout – delete stored secret key with keyserv

SYNOPSIS

/usr/bin/keylogout [**–f**]

AVAILABILITY

SUNWcsu

DESCRIPTION

keylogout deletes the key stored by the key server process **keyserv**(1M). Further access to the key is revoked; however, current session keys may remain valid until they expire or are refreshed.

Deleting the keys stored by **keyserv** will cause any background jobs or scheduled **at**(1) jobs that need secure RPC services to fail. Since only one copy of the key is kept on a machine, it is a bad idea to place a call to this command in your **.logout** file since it will affect other sessions on the same machine.

OPTIONS

–f Force **keylogout** to delete the secret key for the super-user. By default, **keylogout** by the super-user is disallowed because it would break all RPC services, such as NFS, that are started by the super-user.

SEE ALSO

at(1), **chkey**(1), **login**(1), **keylogin**(1), **keyserv**(1M), **newkey**(1M), **publickey**(4)

NAME

kill – terminate a process by default

SYNOPSIS

kill [*–signal*] *pid*. . .
kill –signal *–pgid*. . .
kill –l

AVAILABILITY

SUNWcsu

DESCRIPTION

kill sends a signal to the specified processes. The value of *signal* may be numeric or symbolic. (See **signal**(5)). The symbolic signal name is the name as it appears in **/usr/include/sys/signal.h**, with the **SIG** prefix stripped off. Signal 15 (**SIGTERM**) is sent by default; this will normally kill processes that do not catch or ignore the signal.

pid and *pgid* are unsigned numeric strings that identify which process(es) should receive the signal. If *pid* is used, the process with process ID *pid* is selected. If *pgid* is used, all processes with process group ID *pgid* are selected.

The process number of each asynchronous process started with **&** is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using **ps**(1).

The details of the kill are described in **kill**(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The signalled process must belong to the current user unless the user is the super-user.

OPTIONS

–l **kill** will print a list of symbolic signal names.

The 3 shells, **sh**(1), **csh**(1), and **ksh**(1) each have built-in versions of the **kill** command for processes identified with a jobid number.

SEE ALSO

csh(1), **jobs**(1), **ksh**(1), **ps**(1), **sh**(1), **shell_builtins**(1), **kill**(2), **signal**(3C), **signal**(5)

NOTES

sh

The Bourne shell, **sh**, has a built-in version of **kill** to provide the functionality of the **kill** command for processes identified with a *jobid*. The **sh** syntax is:

kill [*–signal*] **%***jobid*

csh

The C-shell, **csh**, also has a built-in **kill** command, whose syntax is:

kill [*–sig*] [*pid*] [**%***job*] . . .
kill –l

The **csh kill** built-in sends the TERM (terminate) signal, by default, or the signal specified, to the specified process ID, the *job* indicated, or the current *job*. Signals are either given by number or by name. There is no default. Typing **kill** does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

–l List the signal names that can be sent.

ksh

The **ksh kill**'s syntax is:

kill [*–sig*] **%***job* **. . .**
kill –l

The **ksh kill** sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in **signal**(5) stripped of the prefix ‘‘SIG’’ with the exception that SIGCHD is named CHLD). If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument *job* can be the process id of a process that is not a member of one of the active jobs. In the second form, **kill –l**, the signal numbers and names are listed.

NAME

kinit – Kerberos login utility

SYNOPSIS

kinit [**–ilrv**] [*username*]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **kinit** command is used to login to the Kerberos authentication and authorization system. Note that only registered Kerberos users can use the Kerberos system. For information about registering as a Kerberos user, see the **kerberos**(1) manual page.

When you use **kinit** without options, the utility prompts for your *username* and Kerberos password, and tries to authenticate your login with the local Kerberos server. The *username* can be specified on the command line if desired.

If Kerberos authenticates the login attempt, **kinit** retrieves your initial ticket and puts it in the ticket file specified by your **KRBTKFILE** environment variable. If this variable is undefined, your ticket will be stored in the file **/tmp/tkt***uid*, where *uid* specifies your user identification number. Tickets expire after a specified lifetime, after which **kinit** must be run again to refresh the tickets. The default ticket lifetime is 8 hours.

The **kdestroy**(1) command may be used to destroy any active tickets before you end your login session.

OPTIONS

–i	**kinit** prompts you for a Kerberos instance.
–l	**kinit** prompts you for a ticket lifetime in minutes. Due to protocol restrictions in Kerberos Version 4, this value must be between 5 and 1275 minutes.
–r	**kinit** prompts you for a Kerberos realm. This option lets you authenticate yourself with a remote Kerberos server.
–v	Verbose mode. **kinit** prints the name of the ticket file used, and a status message indicating the success or failure of your login attempt.

SEE ALSO

kdestroy(1), **kerberos**(1), **klist**(1)

BUGS

The **–r** option has not been fully implemented.

AUTHORS

Steve Miller, MIT Project Athena / Digital Equipment Corporation
Clifford Neuman, MIT Project Athena

NAME

klist – list currently held Kerberos tickets

SYNOPSIS

klist [–**st**] [–**file** *name*] [–**srvtab**]

AVAILABILITY

SUNWcsu

DESCRIPTION

klist prints the name of the ticket file, the identity of the principal that the tickets are for (as listed in the ticket file), and the principal names of all Kerberos tickets currently held by the user, along with the issue and expire time for each authenticator. Principal names are listed in the form *name***.***instance***@***realm*, with the '.' omitted if the instance is NULL, and the '@' omitted if the realm is NULL.

The value of the KRBTKFILE environment variable is used as the name of the ticket file. If this environment variable is not set, then the file **/tmp/tkt***uid* is used, where *uid* is the current user-id of the user.

OPTIONS

–**s**	Silent. Do not print the issue and expire times, the name of the ticket file, or the identity of the principal.
–**t**	**klist** checks for the existence of a non-expired ticket-granting-ticket in the ticket file. If one is present, it exits with status 0, else it exits with status 1. No output is generated when this option is specified.
–**file** *name*	File *name* is used as the ticket file.
–**srvtab**	The file is treated as a service key file, and the names of the keys contained therein are printed. If no file is specified with a –**file** option, the default is **/etc/srvtab**.

FILES

/etc/krb.conf	to get the name of the local realm
/tmp/tkt*uid*	as the default ticket file
/etc/srvtab	as the default service key file

SEE ALSO

kerberos(1), **kinit**(1), **kdestroy**(1)

BUGS

When reading a file as a service key file, very little sanity or error checking is performed.

NAME

ksh, rksh – KornShell, a standard / restricted command and programming language

SYNOPSIS

ksh [±**aefhikmnoprstuvx**] [±**o** *option*] . . . [−**c** *string*] [*arg* . . .]

rksh [±**aefhikmnoprstuvx**] [±**o** *option*] . . . [−**c** *string*] [*arg* . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

ksh is a command and programming language that executes commands read from a terminal or a file. **rksh** is a restricted version of the command interpreter **ksh**; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See **Invocation** below for the meaning of arguments to the shell.

Definitions

A *metacharacter* is one of the following characters:

; & ( ) | < > new-line space tab

A *blank* is a **tab** or a **space**. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *functions* and *variables*. A *word* is a sequence of *characters* separated by one or more non-quoted *metacharacters*.

A *command* is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A *special command* is a command that is carried out by the shell without creating a separate process. Except for documented side effects, most special commands can be implemented as separate utilities.

Commands

A *simple-command* is a sequence of *blank* separated words which may be preceded by a variable assignment list. (See **Environment** below.) The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see **exec**(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see **signal**(3C) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by |. The standard output of each command but the last is connected by a **pipe**(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more *pipelines* separated by **;**, **&**, **&&**, or ||, and optionally terminated by **;**, **&**, or |**&**. Of these five symbols, **;**, **&**, and |**&** have equal precedence, which is lower than that of **&&** and ||. The symbols **&&** and || also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol |**&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell.

The standard input and output of the spawned command can be written to and read from by the parent shell using the −**p** option of the special commands **read** and **print** described in **Special Commands**. The symbol **&&** (| |) causes the *list* following it to be executed only if the preceding pipeline returns zero (or a non-zero) value. An arbitrary number of new-lines may appear in a *list*, instead of a semicolon, to delimit a command.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *identifier* [**in** *word* . . .] **;do** *list* **;done**

Each time a **for** command is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* . . . is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see **Parameter Substitution** below). Execution ends when there are no more words in the list.

select *identifier* [**in** *word* . . .] **;do** *list***;done**

A **select** command prints to standard error (file descriptor 2), the set of *word*s, each preceded by a number. If **in** *word* . . . is omitted, then the positional parameters are used instead (see **Parameter Substitution** below). The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed *word*s, then the value of the variable *identifier* is set to the *word* corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the variable *identifier* is set to **NULL**. (See **Blank Interpretation** about **NULL**). The contents of the line read from standard input is saved in the shell variable **REPLY**. The *list* is executed for each selection until a **break** or *end-of-file* is encountered. If the **REPLY** variable is set to **NULL** by the execution of *list*, then the selection list is printed before displaying the **PS3** prompt for the next selection.

case *word* **in** [[**(**]*pattern* [| *pattern*] . . . **)** *list* **;;** **]** **. . . esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see **File Name Generation** below).

if *list* **;then** *list* [[**elif** *list* **;** **then** *list*] . . .]**;** **else** *list***;fi**

The *list* following **if** is executed and, if it returns an exit status of zero, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **;do** *list* **;done**
until *list* **;do** *list* **;done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list***)** Execute *list* in a separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{*list***}** *list* is simply executed. Note that unlike the metacharacters **(** and **)**, **{** and **}** are *reserved word*s and must occur at the beginning of a line or after a **;** in order to be recognized.

[[*expression***]]**
Evaluates *expression* and returns a zero exit status when *expression* is true. See **Conditional Expressions** below, for a description of *expression*.

function *identifier* **{** *list* **;}**
*identifier***( ) {** *list* **;}**
Define a function which is referenced by *identifier*. The body of the function is the *list* of commands between **{** and **}**. (See **Functions** below).

time *pipeline*
The *pipeline* is executed and the elapsed time as well as the user and system time are printed to standard error.

The following reserved words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { } function select time [[ ]]

Comments

A word beginning with **#** causes that word and all the following characters up to a newline to be ignored.

Aliasing

The first word of each command is replaced by the text of an **alias** if an **alias** for this word has been defined. An **alias** name consists of any number of characters excluding metacharacters, quoting characters, file expansion characters, parameter and command substitution characters, and **=**. The replacement string can contain any valid shell script including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for aliases. If the last character of the alias value is a *blank* then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the reserved words listed above. Aliases can be created, listed, and exported with the **alias** command and can be removed with the **unalias** command. Exported aliases remain in effect for scripts invoked by name, but must be reinitialized for separate invocations of the shell (see **Invocation** below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the **alias** definition command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the corresponding command is looked up and becomes undefined each time the **PATH** variable is reset. These aliases remain *tracked* so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The **–h** option of the **set** command makes each referenced command name into a tracked alias.

The following *exported aliases* are compiled into (and built-in to) the shell but can be unset or redefined:

autoload=′typeset –fu′
false=′let 0′
functions=′typeset –f′
hash=′alias –t′
history=′fc –l′
integer=′typeset –i′
nohup=′nohup ′
r=′fc –e –′
true=′:′
type=′whence –v′

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, then the word up to a **/** is checked to see if it matches a user name. If a match is found, the ~ and the matched login name are replaced by the login directory of the matched user. This is called a *tilde* substitution. If no match is found, the original text is left unchanged. A ~ by itself, or in front of a **/**, is replaced by **$HOME**. A ~ followed by a **+** or – is replaced by **$PWD** and **$OLDPWD** respectively.

In addition, *tilde* substitution is attempted when the value of a *variable assignment* begins with a ~.

Command Substitution

The standard output from a *command* enclosed in parenthesis preceded by a dollar sign (**$(***command***)**) or a pair of grave accents (**‘‘**) may be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See **Quoting** below.) The command substitution **$(cat** *file***)** can be replaced by the equivalent but faster **(<***file***)**. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

An arithmetic expression enclosed in double parenthesis preceded by a dollar sign (**$((***arithmetic-expression***))**) is replaced by the value of the arithmetic expression within the double parenthesis.

Process Substitution

This feature is available in SunOS and only on versions of the UNIX operating system that support the **/dev/fd** directory for naming open files. Each command argument of the form **<(***list***)** or **>(***list***)** will run process *list* asynchronously connected to some file in **/dev/fd**. The name of this file will become the argument to the command. If the form with **>** is selected then writing on this file will provide input for *list*. If **<** is used, then the file passed as an argument will contain the output of the *list* process. For example,

```
paste <(cut –f1 file1) <(cut –f3 file2) | tee >( process1) >( process2)
```

cuts fields 1 and 3 from the files *file1* and *file2* respectively, **paste**s the results together, and sends it to the processes *process1* and *process2*, as well as putting it onto the standard output. Note that the file, which is passed as an argument to the command, is a UNIX **pipe**(2) so programs that expect to **lseek**(2) on the file will not work.

Parameter Substitution

A *parameter* is an *identifier*, one or more digits, or any of the characters ∗, **@**, **#**, **?**, –, **$**, and **!**. A *variable* (a *parameter* denoted by an *identifier*) has a *value* and zero or more *attributes*. *variable*s can be assigned *value*s and *attribute*s by using the **typeset** special command. The attributes supported by the shell are described later with the **typeset** special command. Exported variables pass values and attributes to the environment.

The shell supports a one-dimensional array facility. An element of an array variable is referenced by a *subscript*. A *subscript* is denoted by a **[**, followed by an *arithmetic expression* (see **Arithmetic Evaluation** below) followed by a **]**. To assign values to an array, use **set –A** *name value* **. . ..** The *value* of all subscripts must be in the range of 0 through 1023. Arrays need not be declared. Any reference to a variable with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the element zero.

The *value* of a *variable* may be assigned by writing:

name=*value* [*name*=*value*] . . .

If the integer attribute, **–i**, is set for *name*, the *value* is subject to arithmetic evaluation as described below.

Positional parameters, parameters denoted by a number, may be assigned values with the **set** special command. Parameter **$0** is set from argument zero when the shell is invoked.

The character **$** is used to introduce substitutable *parameters*.

${*parameter***}**

The shell reads all the characters from **${** to the matching **}** as part of the same word even if it contains braces or metacharacters. The value, if any, of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a variable is subscripted. If *parameter* is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is ∗ or **@**, then all the positional parameters, starting with **$1**, are substituted (separated by a field separator character). If an array *identifier* with subscript ∗ or **@** is used, then the value for each of the elements is substituted (separated by a field separator character).

${#*parameter***}**

If *parameter* is ∗ or **@**, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

${#*identifier***[**∗**]}**

The number of elements in the array *identifier* is substituted.

${*parameter***:–***word***}**
If *parameter* is set and is non-null then substitute its value; otherwise, substitute *word*.

${*parameter***:=***word***}**
If *parameter* is not set or is **NULL** then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**
If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter***:+***word***}**
If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

${*parameter***#***pattern***}**
${*parameter***##***pattern***}**
If the shell *pattern* matches the beginning of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise, the value of this *parameter* is substituted. In the first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted. The result is unspecified when *parameter* is **@**, ∗, or an array variable with subscript **@**, or ∗.

${*parameter***%***pattern***}**
${*parameter***%%***pattern***}**
If the shell *pattern* matches the end of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched part deleted; otherwise, substitute the value of *parameter*. In the first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted. The result is unspecified when *parameter* is **@**, ∗, or an array variable with subscript **@**, or ∗.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is **NULL** (see **Blank Interpretation** for description about **NULL**);

```
echo ${d:–$(pwd)}
```

If the colon (**:**) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
–	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last executed command.
$	The process number of this shell.
_	Initially, the value of **_** is an absolute pathname of the shell or script

being executed as passed in the *environment*. Subsequently it is assigned the last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail.

!	The process number of the last background command invoked.
ERRNO	The value of **errno** as set by the most recently failed system call. This value is system dependent and is intended for debugging purposes.
LINENO	The line number of the current line within the script or function being executed.
OLDPWD	The previous working directory set by the **cd** command.
OPTARG	The value of the last option argument processed by the **getopts** special command.
OPTIND	The index of the last option argument processed by the **getopts** special command.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the **cd** command.
RANDOM	Each time this variable is referenced, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.
REPLY	This variable is set by the **select** statement and by the **read** special command when no arguments are supplied.
SECONDS	Each time this variable is referenced, the number of seconds since shell invocation is returned. If this variable is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following variables are used by the shell:

CDPATH	The search path for the **cd** command.
COLUMNS	If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing **select** lists.
EDITOR	If the value of this variable ends in **emacs**, **gmacs**, or **vi** and the **VISUAL** variable is not set, then the corresponding option (see the **set** special command below) will be turned on.
ENV	If this variable is set, then parameter substitution is performed on the value to generate the pathname of the script that will be executed when the **shell** is invoked. (See **Invocation** below.) This file is typically used for **alias** and **function** definitions.
FCEDIT	The default editor name for the **fc** command.
FPATH	The search path for function definitions. By default the **FPATH**

	directories are searched after the **PATH** variable. If an executable file is found, then it is read and executed in the current environment. **FPATH** is searched before **PATH** when a function with the −**u** attribute is referenced. The preset alias **autoload** preset alias causes a function with the −**u** attribute to be created.
IFS	Internal field separators, normally **space**, **tab**, and **new-line** that are used to separate command words which result from command or parameter substitution and for separating words with the special command **read**. The first character of the **IFS** variable is used to separate arguments for the **$*** substitution (See **Quoting** below).
HISTFILE	If this variable is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See **Command re-entry** below.)
HISTSIZE	If this variable is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is **128**.
HOME	The default argument (home directory) for the **cd** command.
LC_CTYPE	Determines how the shell handles characters. When **LC_CTYPE** is set to a valid value, the shell can display and handle text and filenames containing valid characters for that locale. However, the shell is not multibyte (EUC) capable. In the "C" locale, only ASCII characters are valid. If **LC_CTYPE** (see **environ**(5)) is not set in the environment, the operational behavior of the shell is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale prevails.
LINES	If this variable is set, the value is used to determine the column length for printing **select** lists. Select lists will print vertically until about two-thirds of **LINES** lines are filled.
MAIL	If this variable is set to the name of a mail file *and* the **MAILPATH** variable is not set, then the shell informs the user of arrival of mail in the specified file.
MAILCHECK	This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** variables. The default value is **600** seconds. When the time has elapsed the shell will check before issuing the next prompt.
MAILPATH	A colon (**:**) separated list of file names. If this variable is set, then the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name

	can be followed by a **?** and a message that will be printed. The message will undergo parameter substitution with the variable **$_** defined as the name of the file that has changed. The default message is **you have mail in $_**.
PATH	The search path for commands (see **Execution** below). The user may not change **PATH** if executing under **rksh** (except in **.profile**).
PS1	The value of this variable is expanded for parameter substitution to define the primary prompt string which by default is “**$** ”. The character **!** in the primary prompt string is replaced by the *command* number (see *Command Re-entry* below). Two successive occurrences of **!** will produce a single **!** when the prompt string is printed.
PS2	Secondary prompt string, by default “**>** ”.
PS3	Selection prompt string used within a **select** loop, by default “**#?** ”.
PS4	The value of this variable is expanded for parameter substitution and precedes each line of an execution trace. If omitted, the execution trace prompt is “**+** ”.
SHELL	The pathname of the *shell* is kept in the environment. At invocation, if the basename of this variable is **rsh**, **rksh**, or **krsh**, then the shell becomes restricted.
TMOUT	If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the **PS1** prompt. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)
VISUAL	If the value of this variable ends in **emacs**, **gmacs**, or **vi** then the corresponding option (see Special Command **set** below) will be turned on.

The shell gives default values to **PATH**, **PS1**, **PS2**, **PS3**, **PS4**, **MAILCHECK**, **FCEDIT**, **TMOUT** and **IFS**, while **HOME**, **SHELL ENV** and **MAIL** are not set at all by the shell (although **HOME** *is* set by **login**(1)). On some systems **MAIL** and **SHELL** are also set by **login**(1).

Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit **NULL** arguments ("") or (**’’**) are retained. Implicit NULL arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters *, **?**, and **[** unless the **−f** option has been **set**. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with lexicographically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a *pattern* is used for file name generation, the character period (**.**) at the start of a file name or immediately following a **/**, as well as the character **/** itself, must be matched explicitly. In other instances of pattern matching the **/** and **.** are not treated specially.

*	Matches any string, including the NULL string.
?	Matches any single character.
[. . .**]**	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "! ", then any character not enclosed is matched. A – can be included in the character set by putting it as the first or last character.

A *pattern-list* is a list of one or more patterns separated from each other with a | . Composite patterns can be formed with one or more of the following:

?(*pattern-list***)**
Optionally matches any one of the given patterns.

*(*pattern-list***)**
Matches zero or more occurrences of the given patterns.

+(*pattern-list***)**
Matches one or more occurrences of the given patterns.

@(*pattern-list***)**
Matches exactly one of the given patterns.

!(*pattern-list***)**
Matches anything, except one of the given patterns.

Quoting

Each of the *metacharacters* listed above (See **Definitions** above) has a special meaning to the shell and causes termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\new-line** is removed. All characters enclosed between a pair of single quote marks (′′), are quoted. A single quote cannot appear within single quotes. Inside double quote marks (parameter and command substitution occur and \ quotes the characters \, ‘, ", and **$**. The meaning of **$*** and **$@** is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, **$*** is equivalent to **“$1***d* **$2***d* **. . .”,** where *d* is the first character of the **IFS** variable, whereas **$@** is equivalent to **$1 $2** . . .. Inside grave quote marks (‘‘) \ quotes the characters \, ‘, and **$**. If the grave quotes occur within double quotes, then \ also quotes the character ".

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command **let**. Evaluations are performed using *long* arithmetic. Constants are of the form [*base***#**] *n* where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted then base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression as the C language. All the integral operators, other than **++**, **––**, **?:**, and **,** are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a *variable* can be specified with the –**i** option of the **typeset** special command. Arithmetic evaluation is performed on the value of each assignment to a variable with the –**i** attribute. If you do not specify an arithmetic base, the first assignment to the variable determines the arithmetic base. This base is used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the **let** command is provided. For any command which begins with a **((**, all the characters until a matching **))** are treated as a quoted expression. More precisely, **((**. . .**))** is equivalent to **let** " . . .".

Prompting

When used interactively, the shell prompts with the parameter expanded value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Conditional Expressions

A *conditional expression* is used with the **[[** compound command to test attributes of files and to compare strings. Word splitting and file name generation are not performed on the words between **[[** and **]]**. Each expression can be constructed from one or more of the following unary or binary expressions:

–**a** *file*	True, if *file* exists.
–**b** *file*	True, if *file* exists and is a block special file.
–**c** *file*	True, if *file* exists and is a character special file.
–**d** *file*	True, if *file* exists and is a directory.
–**f** *file*	True, if *file* exists and is an ordinary file.
–**g** *file*	True, if *file* exists and is has its setgid bit set.
–**k** *file*	True, if *file* exists and is has its sticky bit set.
–**n** *string*	True, if length of *string* is non-zero.
–**o** *option*	True, if option named *option* is on.
–**p** *file*	True, if *file* exists and is a fifo special file or a pipe.
–**r** *file*	True, if *file* exists and is readable by current process.
–**s** *file*	True, if *file* exists and has size greater than zero.
–**t** *fildes*	True, if file descriptor number *fildes* is open and associated with a terminal device.
–**u** *file*	True, if *file* exists and has its setuid bit set.
–**w** *file*	True, if *file* exists and is writable by current process.
–**x** *file*	True, if *file* exists and is executable by current process. If *file* exists and is a directory, then the current process has permission to search in the directory.
–**z** *string*	True, if length of *string* is zero.
–**L** *file*	True, if *file* exists and is a symbolic link.
–**O** *file*	True, if *file* exists and is owned by the effective user id of this process.
–**G** *file*	True, if *file* exists and its group matches the effective group id of this process.
–**S** *file*	True, if *file* exists and is a socket.
file1 –**nt** *file2*	True, if *file1* exists and is newer than *file2*.
file1 –**ot** *file2*	True, if *file1* exists and is older than *file2*.

file1 −**ef** *file2*	True, if *file1* and *file2* exist and refer to the same file.
string **=** *pattern*	True, if *string* matches *pattern*.
string **!=** *pattern*	True, if *string* does not match *pattern*.
string1 **<** *string2*	True, if *string1* comes before *string2* based on ASCII value of their characters.
string1 **>** *string2*	True, if *string1* comes after *string2* based on ASCII value of their characters.
exp1 −**eq** *exp2*	True, if *exp1* is equal to *exp2*.
exp1 −**ne** *exp2*	True, if *exp1* is not equal to *exp2*.
exp1 −**lt** *exp2*	True, if *exp1* is less than *exp2*.
exp1 −**gt** *exp2*	True, if *exp1* is greater than *exp2*.
exp1 −**le** *exp2*	True, if *exp1* is less than or equal to *exp2*.
exp1 −**ge** *exp2*	True, if *exp1* is greater than or equal to *exp2*.

In each of the above expressions, if *file* is of the form **/dev/fd/***n*, where *n* is an integer, then the test is applied to the open file whose descriptor number is *n*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

(*expression***)**	True, if *expression* is true. Used to group expressions.
! *expression*	True if *expression* is false.
expression1 **&&** *expression2*	True, if *expression1* and *expression2* are both true.
expression1 || *expression2*	True, if either *expression1* or *expression2* is true.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command. Command and parameter substitution occur before *word* or *digit* is used except as noted below. File name generation occurs only if the pattern matches a single file, and blank interpretation is not performed.

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created. If the file exists, and the **noclobber** option is on, this causes an error; otherwise, it is truncated to zero length.
>**|** *word*	Sames as >, except that it overrides the **noclobber** option.
>>*word*	Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<>*word*	Open file *word* for reading and writing as standard input.
<< [–]*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is

quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occur, **\newline** is ignored, and \ must be used to quote the characters \, **$**, **‘**, and the first character of *word*. If – is appended to **<<**, then all leading tabs are stripped from *word* and from the document.

<&*digit*	The standard input is duplicated from file descriptor *digit* (see **dup**(2)). Similarly for the standard output using **>&***digit*.
<&–	The standard input is closed. Similarly for the standard output using **>&–**.
<&p	The input from the co-process is moved to standard input.
>&p	The output to the co-process is moved to standard output.

If one of the above is preceded by a digit, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

```
... 1>fname 2>&1
```

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *fname*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by **&** and job control is not active, then the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see **environ**(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a variable for each name found, giving it the corresponding value and marking it *export*. Executed commands inherit the environment. If the user modifies the values of these variables or creates new ones, using the **export** or **typeset –x** commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in **export** or **typeset –x** commands.

The environment for any *simple-command* or *function* may be augmented by prefixing it with one or more variable assignments. A variable assignment argument is a word of the form *identifier=value*. Thus:

TERM=450 *cmd args*
and
(export TERM; TERM=450; *cmd args)*

are equivalent (as far as the above execution of *cmd* is concerned except for special commands listed below that are preceded with a dagger).

If the –**k** flag is set, *all* variable assignment arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set –k
echo a=b c
```

This feature is intended for use with scripts written for early versions of the shell and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions

The **function** reserved word, described in the **Commands** section above, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See **Execution** below.)

Functions execute in the same process as the caller and share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on **EXIT** set inside a function is executed after the function completes in the environment of the caller. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

The names of all functions can be listed with **typeset +f**. **typeset** –**f** lists all function names as well as the text of all functions. **typeset** –**f** *function-names* lists the text of the named functions only. Functions can be undefined with the –**f** option of the **unset** special command.

Ordinarily, functions are **unset** when the shell executes a shell script. The –**xf** option of the **typeset** command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be specified in the **ENV** file with the –**xf** option of **typeset**.

Jobs

If the **monitor** option of the **set** command is turned on, an interactive shell associates a **job** with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the **job**, which was started asynchronously, was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key **ˆZ** (CTRL-Z) which sends a **STOP** signal to the current job. The shell will then normally indicate that the job has been **'Stopped'**, and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A **ˆZ** takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command **''stty tostop''**. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to **job**s in the shell. A **job** can be referred to by the process id of any process of the **job** or by one of the following:

%*number*	The job with the given number.
%*string*	Any job whose command line begins with *string*.
%?*string*	Any job whose command line contains *string*.
%%	Current job.
%+	Equivalent to **%%**.
%–	Previous job.

The shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

When the monitor mode is on, each background job that completes triggers any trap set for **CHLD**.

When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped(running) jobs.' You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by **&** and the **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the **trap** special command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the **Special Commands** listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the **function** call. When the **function** completes or issues a **return**, the positional parameter list is restored and any trap set on **EXIT** within the function is executed. The value of a **function** is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a **special command** or a user defined **function**, a process is created and an attempt is made to execute the command via **exec**(2).

The shell variable **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **/bin:/usr/bin:** (specifying **/bin**, **/usr/bin**, and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a **/** then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and variables are removed in this case. A parenthesized command is executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a **history** file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or if the file it names is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to **fc** then the value of the variable **FCEDIT** is used. If **FCEDIT** is not defined then **/bin/ed** is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name – is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old*=*new* can be used to modify the command before execution. For example, if **r** is aliased to **′fc –e –′** then typing **‘r bad=good c’** will re-execute the most recent command which starts with the letter **c**, replacing the first occurrence of the string **bad** with the string **good**.

In-line Editing Option

Normally, each command line entered from a terminal device is simply typed followed by a new-line (RETURN or LINE FEED). If either the **emacs**, **gmacs**, or **vi** option is active, the user can edit the command line. To be in either of these edit modes **set** the corresponding option. An editing option is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these option names.

The editing features require that the user’s terminal accept RETURN as carriage return without line feed and that a space (‘ ’) must overwrite the current character on the screen.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of **COLUMNS** if it is defined, otherwise 80. If the window width is too small to display the prompt and leave at least 8 columns to enter input, the prompt is truncated from the left. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a **>** if the line extends on the right side of the window, **<** if the line extends on the left, and * if the line extends on both sides of the window.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading ^ in the string restricts the match to begin at the first character in the line.

emacs Editing Mode

This mode is entered by enabling either the **emacs** or **gmacs** option. The only difference between these two modes is the way they handle **^T**. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, **^F** is the notation for control **F**. This is entered by depressing 'f' while holding down the CTRL (control) key. The SHIFT key is *not* depressed. (The notation **^?** indicates the DEL (delete) key.)

The notation for escape sequences is **M-** followed by a character. For example, **M-f** (pronounced Meta f) is entered by depressing ESC (ascii **033**) followed by 'f'. (**M-F** would be the notation for ESC followed by SHIFT (capital) 'F'.)

All edit commands operate from any place on the line (not just at the beginning). Neither the RETURN nor the LINE FEED key is entered after edit commands except when noted.

^F	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The **emacs** editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]*char*	Move cursor forward to character *char* on current line.
M-^]*char*	Move cursor backward to character *char* on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the **stty**(1) command, usually **^H** or **#**.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is **^?** (DEL, the default) then this command will not work).
^T	Transpose current character with next character in **emacs** mode. Transpose two previous characters in **gmacs** mode.
^C	Capitalize current character.
M-c	Capitalize current word.
M-l	Change the current word to lower case.
^K	Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less than the current cursor position, then delete from given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from cursor up to given cursor position.

ˆW	Kill from the cursor to the mark.
M-p	Push the region from the cursor to the mark on the stack.
kill	(User defined kill character as defined by the **stty**(1) command, usually **ˆG** or **@**.) Kill the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
ˆY	Restore last item removed from line. (Yank item back to the line.)
ˆL	Line feed and print current line.
ˆ@	(NULL character) Set mark.
M-*space*	(Meta space) Set mark.
ˆJ	(New line) Execute the current line.
ˆM	(Return) Execute the current line.
eof	End-of-file character, normally **ˆD**, is processed as an End-of-file only if the current line is null.
ˆP	Fetch previous command. Each time **ˆP** is entered the previous command back in time is accessed. Moves back one line when not on the first line of a multi-line command.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.
ˆN	Fetch next command line. Each time **ˆN** is entered the next command line forward in time is accessed.
ˆR*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *string* is terminated by a RETURN or NEW LINE. If string is preceded by a ˆ, the matched line must begin with *string*. If *string* is omitted, then the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.
ˆO	Operate – Execute the current line and fetch the next line relative to current line from the history file.
M-*digits*	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are **ˆF**, **ˆB**, *erase*, **ˆC**, **ˆD**, **ˆK**, **ˆR**, **ˆP**, **ˆN**, **ˆ]**, **M-.**, **M-ˆ]**, **M-_**, **M-b**, **M-c**, **M-d**, **M-f**, **M-h**, **M-l** and **M-ˆH**.
M-*letter*	Soft-key – Your alias list is searched for an alias by the name **_***letter* and if an alias of this name is defined, its value will be inserted on the input queue. The *letter* must not be one of the above meta-functions.
M-[*letter*	Soft-key – Your alias list is searched for an alias by the name **__***letter* and if an alias of this name is defined, its value will be inserted on the input queue. The can be used to program functions keys on many terminals.
M-.	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_	Same as **M-.**.
M-*	Attempt file name generation on the current word. An asterisk is appended if the word doesn't match any file or contain any special

	pattern characters.
M-ESC	File name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a **/** is appended if the file is a directory and a space is appended if the file is not a directory.
M-=	List files matching current word pattern if an asterisk were appended.
ˆU	Multiply parameter of next command by 4.
\	Escape next character. Editing characters, the user's erase, kill and interrupt (normally **ˆ?**) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
ˆV	Display version of the shell.
M-#	Insert a **#** at the beginning of the line and execute it. This causes a comment to be inserted in the history file.

vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the *input* mode. To edit, the user enters *control* mode by typing ESC (**033**) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* prior to the command.

When in **vi** mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the option **viraw** is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase	(User defined erase character as defined by the **stty**(1) command, usually **ˆH** or **#**.) Delete previous character.
ˆW	Delete the previous blank separated word.
ˆD	Terminate the shell.
ˆV	Escape next character. Editing characters and the user's erase or kill characters may be entered in a command line or in a search string if preceded by a **ˆV**. The **ˆV** removes the next character's editing features (if any).
\	Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands will move the cursor.

[*count*]**l**	Cursor forward (right) one character.
[*count*]**w**	Cursor forward one alpha-numeric word.
[*count*]**W**	Cursor to the beginning of the next word that follows a blank.
[*count*]**e**	Cursor to end of word.
[*count*]**E**	Cursor to end of the current blank delimited word.
[*count*]**h**	Cursor backward (left) one character.
[*count*]**b**	Cursor backward one word.
[*count*]**B**	Cursor to preceding blank separated word.
[*count*]**|**	Cursor to column *count*.
[*count*]**f***c*	Find the next character *c* in the current line.
[*count*]**F***c*	Find the previous character *c* in the current line.
[*count*]**t***c*	Equivalent to **f** followed by **h**.
[*count*]**T***c*	Equivalent to **F** followed by **l**.
[*count*]**;**	Repeats *count* times, the last single character find command, **f**, **F**, **t**, or **T**.
[*count*]**,**	Reverses the last single character find command *count* times.
0	Cursor to start of line.
^	Cursor to first non-blank character in line.
$	Cursor to end of line.
%	Moves to balancing **(**, **)**, **{**, **}**, **[**, or **]**. If cursor is not on one of the above characters, the remainder of the line is searched for the first occurrence of one of the above characters first.

Search Edit Commands

These commands access your command history.

[*count*]**k**	Fetch previous command. Each time **k** is entered the previous command back in time is accessed.
[*count*]**–**	Equivalent to **k**.
[*count*]**j**	Fetch next command. Each time **j** is entered the next command forward in time is accessed.
[*count*]**+**	Equivalent to **j**.
[*count*]**G**	The command number *count* is fetched. The default is the least recent history command.
/*string*	Search backward through history for a previous command containing *string*. *string* is terminated by a RETURN or NEW LINE. If *string* is preceded by a **^**, the matched line must begin with *string*. If *string* is NULL the previous string will be used.
?*string*	Same as **/** except that search will be in the forward direction.
n	Search for next match of the last pattern to **/** or **?** commands.

N Search for next match of the last pattern to **/** or **?**, but in reverse direction. Search history for the *string* entered by the previous **/** command.

Text Modification Edit Commands

These commands will modify the line.

a Enter input mode and enter text after the current character.

A Append text to the end of the line. Equivalent to **$a**.

[*count*]**c***motion*

c[*count*]*motion*
Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is **c**, the entire line will be deleted and input mode entered.

C Delete the current character through the end of line and enter input mode. Equivalent to **c$**.

[*count*]**s** Delete *count* characters and enter input mode.

S Equivalent to **cc**.

D Delete the current character through the end of line. Equivalent to **d$**.

[*count*]**d***motion*

d[*count*]*motion*
Delete current character through the character that *motion* would move to. If *motion* is **d ,** the entire line will be deleted.

i Enter input mode and insert text before the current character.

I Insert text before the beginning of the line. Equivalent to **0i**.

[*count*]**P** Place the previous text modification before the cursor.

[*count*]**p** Place the previous text modification after the cursor.

R Enter input mode and replace characters on the screen with characters you type overlay fashion.

[*count*]**r***c* Replace the *count* character(s) starting at the current cursor position with *c*, and advance the cursor.

[*count*]**x** Delete current character.

[*count*]**X** Delete preceding character.

[*count*]**.** Repeat the previous text modification command.

[*count*]~ Invert the case of the *count* character(s) starting at the current cursor position and advance the cursor.

[*count*]**_** Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.

* Causes an * to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

\\ Filename completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a **/** is appended if the file is a directory and a space is appended if the file is not a directory.

Other Edit Commands

Miscellaneous commands.

[*count*]**y***motion*

y[*count*]*motion*
Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.

Y Yanks from current position to end of line. Equivalent to **y$**.

u Undo the last text modifying command.

U Undo all the text modifying commands performed on the line.

[*count*]**v** Returns the command **fc −e ${VISUAL:−${EDITOR:−vi}}** *count* in the input buffer. If *count* is omitted, then the current line is used.

ˆL Line feed and print current line. Has effect only in control mode.

ˆJ (New line) Execute the current line, regardless of mode.

ˆM (Return) Execute the current line, regardless of mode.

If the first character of the command is a **#**, then this command deletes this **#** and each **#** that follows a newline. Otherwise, sends the line after inserting a **#** in front of each line in the command. Useful for causing the current line to be inserted in the history as a comment and removing comments from previous comment commands in the history file.

= List the file names that match the current word if an asterisk were appended it.

@*letter* Your alias list is searched for an alias by the name **_***letter* and if an alias of this name is defined, its value will be inserted on the input queue for processing.

Special Commands

The following *simple-commands* are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is zero. Commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

† **:** [*arg* . . .]

The command only expands parameters.

† **.** *file* **[** *arg* **. . .]**

Read the complete *file* then execute the commands. The commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed.

†† **alias** [**–tx**] [*name*[**=***value*]] . . .

alias with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **–t** flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the **–t** flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **–x** flag is used to set or print *exported alias*es. An *exported alias* is defined for scripts invoked by name. The exit status is non-zero if a *name* is given, but no value, and no alias has been defined for the *name*.

bg [**%***job***. . .]**

This command is only on systems that support job control. Puts each specified *job* into the background. The current job is put in the background if *job* is not specified. See **Jobs** for a description of the format of *job*.

† **break** [*n*]

Exit from the enclosed **for**, **while**, **until**, or **select** loop, if any. If *n* is specified then **break** *n* levels.

† **continue** [*n*]

Resume the next iteration of the enclosed **for**, **while**, **until**, or **select** loop. If *n* is specified then resume at the *n*-th enclosed loop.

cd [*arg*]
cd *old new*

This command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is – the directory is changed to the previous directory. The shell variable **HOME** is the default *arg*. The variable **PWD** is set to the current directory. The shell variable **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is null (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a **/** then the search path is not used. Otherwise, each directory in the path is searched for *arg*.

The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.

The **cd** command may not be executed by **rksh**.

echo [*arg* . . .]
See **echo**(1) for usage and description.

† **eval** [*arg* . . .]
The arguments are read as input to the shell and the resulting command(s) executed.

† **exec** [*arg* . . .]
If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input / output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input / output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

† **exit** [*n*]
Causes the shell to exit with the exit status specified by *n*. The value will be the least significant 8 bits of the specified status. If *n* is omitted then the exit status is that of the last command executed. When **exit** occurs when executing a trap, the last command refers to the command that executed before the trap was invoked. An end-of-file will also cause the shell to exit except for a shell which has the **ignoreeof** option (See **set** below) turned on.

†† **export** [*name*[=*value*]] . . .
The given *name*s are marked for automatic export to the **environment** of subsequently-executed commands.

fc [–**e** *ename*] –**nlr**] [*first* [*last*]]
fc –**e** – [*old*=*new*] [*command*]
In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the –**l** flag is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the variable **FCEDIT** (default **/bin/ed**) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and –16 for listing. The flag –**r** reverses the order of the commands and the flag –**n** suppresses command numbers when listing. In the second form the *command* is re-executed after the substitution *old*=*new* is performed.

fg [**%***job***. . .]**
This command is only on systems that support job control. Each *job* specified is brought to the foreground. Otherwise, the current job is brought into the foreground. See **Jobs** for a description of the format of *job*.

getopts *optstring name* [*arg* . . .]

Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a **+** or a –. An option not beginning with **+** or – or the argument – – ends the options. *optstring* contains the letters that **getopts** recognizes. If a letter is followed by a **:**, that option is expected to have an argument. The options can be separated from the argument by blanks.

getopts places the next option letter it finds inside variable *name* each time it is invoked with a **+** prepended when *arg* begins with a **+**. The index of the next *arg* is stored in **OPTIND**. The option argument, if any, gets stored in **OPTARG**.

A leading **:** in *optstring* causes **getopts** to store the letter of an invalid option in **OPTARG**, and to set *name* to **?** for an unknown option and to **:** when a required option is missing. Otherwise, **getopts** prints an error message. The exit status is non-zero when there are no more options. See **getoptcvt**(1) for usage and description.

jobs [–**lnp**] [**%***job* **. . .]**

Lists information about each given job; or all active jobs if *job* is omitted. The –**l** flag lists process ids in addition to the normal information. The –**n** flag only displays jobs that have stopped or exited since last notified. The –**p** flag causes only the process group to be listed. See **Jobs** for a description of the format of *job*.

kill [–*sig*] **%***job* **. . .**

kill –**l**

Sends either the **TERM** (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in **signal**(5) stripped of the prefix ‘‘SIG’’ with the exception that **SIGCHD** is named **CHLD**). If the signal being sent is **TERM** (terminate) or **HUP** (hangup), then the job or process will be sent a **CONT** (continue) signal if it is stopped. The argument *job* can be the process id of a process that is not a member of one of the active jobs. See **Jobs** for a description of the format of *job*. In the second form, **kill** –**l**, the signal numbers and names are listed.

let *arg* **. . .**

Each *arg* is a separate *arithmetic expression* to be evaluated. See **Arithmetic Evaluation** above, for a description of arithmetic expression evaluation.

The exit status is **0** if the value of the last expression is non-zero, and **1** otherwise.

† **newgrp** [*arg* . . .]

Equivalent to **exec /bin/newgrp** *arg* . . ..

print [–**Rnprsu**[*n*]] [*arg* . . .]

The shell output mechanism. With no flags or with flag – or – –, the arguments are printed on standard output as described by **echo**(1). The exit status is **0**, unless the output file is not open for writing.

–**n** suppresses **new-line** from being added to the output.

–**R**

–**r** (raw mode) ignore the escape conventions of **echo**. The –**R** option will print all subsequent arguments and options other than **–n**.

–p causes the arguments to be written onto the pipe of the process spawned with **|&** instead of standard output.

–s causes the arguments to be written onto the history file instead of standard output.

–u [*n*] flag can be used to specify a one digit file descriptor unit number *n* on which the output will be placed. The default is **1**.

pwd Equivalent to **print –r – $PWD print –r – $PWD**

read [**–prsu**[*n*]] [*name***?***prompt*] [*name* . . .]

The shell input mechanism. One line is read and is broken up into fields using the characters in **IFS** as separators. The escape character, **(\)**, is used to remove any special meaning for the next character and for line continuation. In raw mode, **–r,** the \ character is not treated specially. The first field is assigned to the first *name*, the second field to the second *name*, etc., with leftover fields assigned to the last *name*. The **–p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **–s** flag is present, the input will be saved as a command in the history file. The flag **–u** can be used to specify a one digit file descriptor unit *n* to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is **0**. If *name* is omitted then **REPLY** is used as the default *name*. The exit status is **0** unless the input file is not open for reading or an end-of-file is encountered. An end-of-file with the **–p** option causes cleanup for this process so that another can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* on standard error when the shell is interactive. The exit status is **0** unless an end-of-file is encountered.

†† **readonly** [*name*[**=***value*]] . . .

The given *name*s are marked **readonly** and these names cannot be changed by subsequent assignment.

† **return** [*n*]

Causes a shell function or **'.'** script to return to the invoking script with the return status specified by *n*. The value will be the least significant 8 bits of the specified status. If *n* is omitted then the return status is that of the last command executed. If **return** is invoked while not in a function or a **'.'** script, then it is the same as an **exit**.

set [±**aefhkmnopstuvx**] [±**o** *option*]. . . [±**A** *name*] [*arg* . . .]

The flags for this command have meaning as follows:

–A Array assignment. Unset the variable *name* and assign values sequentially from the list *arg*. If **+A** is used, the variable *name* is not unset first.

–a All subsequent variables that are defined are automatically exported.

–e If a command has a non-zero exit status, execute the **ERR** trap, if set, and exit. This mode is disabled while reading profiles.

–f Disables file name generation.

–h	Each command becomes a tracked alias when first encountered.
–k	All variable assignment arguments are placed in the environment for a command, not just those that precede the command name.
–m	Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.
–n	Read commands and check them for syntax errors, but do not execute them. Ignored for interactive shells.
–o	The following argument can be one of the following option names:

allexport	Same as **–a**.
errexit	Same as **–e**.
bgnice	All background jobs are run at a lower priority. This is the default mode.
emacs	Puts you in an **emacs** style in-line editor for command entry.
gmacs	Puts you in a **gmacs** style in-line editor for command entry.
ignoreeof	The shell will not exit on end-of-file. The command **exit** must be used.
keyword	Same as **–k**.
markdirs	All directory names resulting from file name generation have a trailing **/** appended.
monitor	Same as **–m**.
noclobber	Prevents redirection **>** from truncating existing files. Require **>|** to truncate a file when turned on.
noexec	Same as **–n**.
noglob	Same as **–f**.
nolog	Do not save function definitions in history file.
nounset	Same as **–u**.
privileged	Same as **–p**.
verbose	Same as **–v**.
trackall	Same as **–h**.
vi	Puts you in insert mode of a **vi** style in-line editor until you hit escape character **033**. This puts you in control mode. A return

sends the line.

viraw Each character is processed as it is typed in **vi** mode.

xtrace Same as –**x**.

If no option name is supplied then the current
option settings are printed.

–**p** Disables processing of the **$HOME/.profile** file and uses the file **/etc/suid_profile** instead of the **ENV** file. This mode is on whenever the effective uid is not equal to the real uid, or when the effective gid is not equal to the real gid. Turning this off causes the effective uid and gid to be set to the real uid and gid.

–**s** Sort the positional parameters lexicographically.

–**t** Exit after reading and executing one command.

–**u** Treat unset parameters as an error when substituting.

–**v** Print shell input lines as they are read.

–**x** Print commands and their arguments as they are executed.

– Turns off –**x** and –**v** flags and stops examining arguments for flags.

– – Do not change any of the flags; useful in setting **$1** to a value beginning with –. If no arguments follow this flag then the positional parameters are unset.

Using **+** rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**. Unless –**A** is specified, the remaining arguments are positional parameters and are assigned, in order, to **$1 $2** . . .. If no arguments are given then the names and values of all variables are printed on the standard output.

† **shift** [*n*]

The positional parameters from **$***n***+1 $***n***+1 . . .** are renamed **$1 . . .**, default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

stop %*jobid* . . .
stop *pid* . . .

stop stops the execution of a background job(s) by using its *jobid*, or of any process by using its *pid*. (see **ps**(1)).

suspend

Stops the execution of the current shell (but not if it is the login shell).

† **times** Print the accumulated user and system times for the shell and for processes run from the shell.

† **trap** [*arg*] [*sig*] . . .

arg is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. **trap** commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is omitted or is –, then the trap(s) for each *sig* are reset to their original values. If *arg* is the NULL (the empty string, e.g., "") string then this signal is ignored by the shell and by the commands it invokes. If *sig* is **ERR** then *arg* will be executed whenever a command has a non-zero exit status. If *sig* is **DEBUG** then *arg* will be executed after each command. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

†† **typeset** [±**HLRZfilrtux**[*n*]] [*name*[**=***value*]] . . .

Sets attributes and values for shell variables and functions. When **typeset** is invoked inside a function, a new instance of the variables *name* is created. The variables *value* and *type* are restored when the function completes. The following list of attributes may be specified:

–**H** This flag provides UNIX to host-name file mapping on non-UNIX machines.

–**L** Left justify and remove leading blanks from *value*. If *n* is non-zero it defines the width of the field; otherwise, it is determined by the width of the value of first assignment. When the variable is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the –**Z** flag is also set. The –**R** flag is turned off.

–**R** Right justify and fill with leading blanks. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left filled with blanks or truncated from the end if the variable is reassigned. The –**L** flag is turned off.

–**Z** Right justify and fill with leading zeros if the first non-blank character is a digit and the –**L** flag has not been set. If *n* is non-zero it defines the width of the field; otherwise, it is determined by the width of the value of first assignment.

–**f** The names refer to function names rather than variable names. No assignments can be made and the only other valid flags are –**t**, –**u** and –**x**. The flag –**t** turns on execution tracing for this function. The flag –**u** causes this function to be marked undefined. The **FPATH** variable will be searched to find the function definition when the function is referenced. The flag –**x** allows the function definition to remain in effect across shell procedures invoked by name.

–**i** Parameter is an integer. This makes arithmetic faster. If *n* is non-zero it defines the output arithmetic base; otherwise, the first assignment determines the output base.

–**l** All upper-case characters are converted to lower-case. The upper-case flag, –**u** is turned off.

–**r** The given *name*s are marked **readonly** and these names cannot be changed by subsequent assignment.

–t	Tags the variables. Tags are user definable and have no special meaning to the shell.
–u	All lower-case characters are converted to upper-case characters. The lower-case flag, **–l** is turned off.
–x	The given *name*s are marked for automatic export to the **environment** of subsequently-executed commands.

The –**i** attribute can not be specified along with –**R**, –**L**, –**Z**, or –**f**.

Using **+** rather than – causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of *names* (and optionally the *values*) of the *variables* which have these flags set is printed. (Using **+** rather than – keeps the values from being printed.) If no *name*s and flags are given, the *names* and *attributes* of all *variables* are printed.

ulimit [–**HSacdfmnpstv**] [*limit*]

Set or display a resource limit. The available resources limits are listed below. Many systems do not contain one or more of these limits. The limit for a specified resource is set when *limit* is specified. The value of *limit* can be a number in the unit specified below with each resource, or the value **unlimited**. The **H** and **S** flags specify whether the hard limit or the soft limit for the given resource is set. A hard limit cannot be increased once it is set. A soft limit can be increased up to the value of the hard limit. If neither the **H** or **S** options is specified, the limit applies to both. The current resource limit is printed when *limit* is omitted. In this case the soft limit is printed unless **H** is specified. When more that one resource is specified, then the limit name and unit is printed before the value.

–a	Lists all of the current resource limits.
–c	The number of 512-byte blocks on the size of core dumps.
–d	The number of K-bytes on the size of the data area.
–f	The number of 512-byte blocks on files written by child processes (files of any size may be read).
–m	The number of K-bytes on the size of physical memory.
–n	The number of file descriptors plus 1.
–p	The number of 512-byte blocks for pipe buffering.
–s	The number of K-bytes on the size of the stack area.
–t	The number of seconds to be used by each process.
–v	The number of K-bytes for virtual memory.

If no option is given, –**f** is assumed.

umask [*mask*]

The user file-creation mask is set to *mask* (see **umask**(2)). *mask* can either be an octal number or a symbolic value as described in **chmod**(1). If a symbolic value is given, the new **umask** value is the complement of the result of applying *mask* to the complement of the previous umask value. If *mask* is omitted, the current value of the mask is printed.

unalias *name***...**

The *alias*es given by the list of *name*s are removed from the *alias* list.

unset [–**f**] *name* . . .

The variables given by the list of *name*s are unassigned, i.e., their values and attributes are erased. **readonly** variables cannot be unset. If the –**f**, flag is set, then the names refer to *function* names. Unsetting **ERRNO**, **LINENO**, **MAILCHECK**, **OPTARG**, **OPTIND**, **RANDOM**, **SECONDS**, **TMOUT**, and **_** removes their special meaning even if they are subsequently assigned to.

† **wait** [*job*]

Wait for the specified *job* and report its termination status. If *job* is not given then all currently active child processes are waited for. The exit status from this command is that of the process waited for. See **Jobs** for a description of the format of *job*.

whence [–**pv**] *name* . . .

For each *name*, indicate how it would be interpreted if used as a command name.

The –**v** flag produces a more verbose report.

The –**p** flag does a path search for *name* even if name is an alias, a function, or a reserved word.

Invocation

If the shell is invoked by **exec**(2), and the first character of argument zero (**$0**) is –, then the shell is assumed to be a **login** shell and commands are read from **/etc/profile** and then from either **.profile** in the current directory or **$HOME/.profile**, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment variable **ENV** if the file exists. If the –**s** flag is not present and *arg* is, then a path search is performed on the first *arg* to determine the name of the script to execute. The script *arg* must have read permission and any **setuid** and **setgid** settings will be ignored. If the script is not found on the path, *arg* is processed as if it named a builtin command or function. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

–**c** *string*	If the –**c** flag is present then commands are read from *string*.
–**s**	If the –**s** flag is present or if no arguments remain then commands are read from the standard input. Shell output, except for the output of the **Special Commands** listed above, is written to file descriptor 2.
–**i**	If the –**i** flag is present or if the shell input and output are attached to a terminal (as told by **ioctl**(2)) then this shell is *interactive*. In this case TERM is ignored (so that **kill 0** does not kill an interactive shell) and INTR is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.
–**r**	If the –**r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

rksh Only

rksh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of **rksh** are identical to those of **ksh**, except that the following are disallowed:

- changing directory (see **cd**(1))
- setting the value of **SHELL**, **ENV**, or **PATH**
- specifying path or command names containing **/**
- redirecting output (**>**, **>|**, **<>**, and **>>**)
- changing group (see **newgrp**(1M)).

The restrictions above are enforced after **.profile** and the **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, **rksh** invokes **ksh** to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by **rksh**.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above). If the shell is being used non-interactively then execution of the shell file is abandoned. Run time errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, then the line number is also printed in square brackets (**[]**) after the command or function name.

FILES

/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh∗
/dev/null

SEE ALSO

cat(1), **chmod**(1), **cut**(1), **echo**(1), **env**(1), **getoptcvt**(1), **paste**(1), **ps**(1), **shell_builtins**(1), **stty**(1), **vi**(1), **newgrp**(1M), **dup**(2), **exec**(2), **fork**(2), **ioctl**(2), **lseek**(2), **pipe**(2), **umask**(2), **ulimit**(2), **wait**(2), **rand**(3C), **signal**(3C), **a.out**(4), **profile**(4), **environ**(5), **signal**(5)

Morris I. Bolsky and David G. Korn, *The KornShell Command and Programming Language*, Prentice Hall, 1989.

NOTES

If a command which is a *tracked alias* is executed, and then a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to **exec** the original command. Use the **–t** option of the **alias** command to correct this situation.

Some very old shell scripts contain a **^** as a synonym for the pipe character **|** .

Using the **fc** built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command **.** *file* reads the whole file before any commands are executed. Therefore, **alias** and **unalias** commands in the file will not apply to any functions defined in the file.

Traps are not processed while a job is waiting for a foreground process. Thus, a trap on **CHLD** won't be executed until the foreground job terminates.

NAME

ksrvtgt – fetch and store Kerberos ticket-granting ticket using a service key

SYNOPSIS

/usr/bin/ksrvtgt *name instance* [[*realm*] *srvtab*]

AVAILABILITY

SUNWcsu

DESCRIPTION

ksrvtgt retrieves a ticket-granting ticket with a lifetime of five minutes for the principal *name***.***instance@realm* (or *name***.***instance@localrealm* if *realm* is not supplied on the command line), decrypts the response using the service key found in the file *srvtab* (or in **/etc/srvtab** if *srvtab* is not specified on the command line), and stores the ticket in the standard ticket cache.

This command is intended primarily for use in shell scripts and other batch-type facilities.

DIAGNOSTICS

Generic kerberos failure (kfailure) can indicate a whole range of problems, the most common of which is the inability to read the service key file.

FILES

/etc/krb.conf	to get the name of the local realm.
/tmp/tkt*uid*	The default ticket file.
/etc/srvtab	The default service key file.

SEE ALSO

kdestroy(1), **kerberos**(1), **kinit**(1), **klist**(1)

NAME

last – display login and logout information about users and terminals

SYNOPSIS

last [–**n** *number* | –*number*] [–**f** *filename*] [*name* | *tty*] . . .

AVAILABILITY

SUNWesu

DESCRIPTION

The **last** command looks in the **/var/adm/wtmpx**, file which records all logins and logouts, for information about a user, a terminal or any group of users and terminals. Arguments specify names of users or terminals of interest. Names of terminals may be given fully or abbreviated. For example **last 10** is the same as **last term/10**. If multiple arguments are given, the information which applies to any of the arguments is printed. For example **last root console** lists all of root's sessions as well as all sessions on the console terminal. **last** displays the sessions of the specified users and terminals, most recent first, indicating the times at which the session began, the duration of the session, and the terminal which the session took place on. If the session is still continuing or was cut short by a reboot, **last** so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

last with no arguments displays a record of all logins and logouts, in reverse order.

If **last** is interrupted, it indicates how far the search has progressed in **/var/adm/wtmpx**. If interrupted with a quit signal (generated by a CTRL-\) **last** indicates how far the search has progressed so far, and the search continues.

OPTIONS

–**n** *number* | –*number*	Limit the number of entries displayed to that specified by *number*. These options are identical; the –*number* option is provided as a transition tool only and will be removed in future releases.
–**f** *filename*	Use *filename* as the name of the accounting file instead of **/var/adm/wtmpx**.

FILES

/var/adm/wtmpx	accounting file

ENVIRONMENT

Date and time format is based on locale specified by the **LC_ALL**, **LC_TIME** or **LANG** environments, in that order of priority.

SEE ALSO

utmp(4)

NAME

lastcomm – display the last commands executed, in reverse order

SYNOPSIS

lastcomm [*command-name*] . . . [*user-name*] . . . [*terminal-name*] . . .

AVAILABILITY

SUNWesu

DESCRIPTION

The **lastcomm** command gives information on previously executed commands. **lastcomm** with no arguments displays information about all the commands recorded during the current accounting file's lifetime. If called with arguments, **lastcomm** only displays accounting entries with a matching *command-name*, *user-name*, or *terminal-name*.

If *terminal-name* is `**- -**´ there was no controlling TTY for the process. The process was probably executed during boot time. If *terminal-name* is `**??**´, the controlling TTY could not be decoded into a printable name.

EXAMPLES

The command:

```
example% lastcomm a.out root term/01
```

produces a listing of all the executions of commands named **a.out**, by user **root** while using the terminal **term/01**.

The command:

```
example% lastcomm root
```

produces a listing of all the commands executed by user **root**.

For each process entry, **lastcomm** displays the following items of information:

- The command name under which the process was called.
- One or more flags indicating special information about the process. The flags have the following meanings:
  - **F** The process performed a **fork** but not an **exec**.
  - **S** The process ran as a set-user-id program.
- The name of the user who ran the process.
- The terminal which the user was logged in on at the time (if applicable).
- The amount of CPU time used by the process (in seconds).
- The date and time the process exited.

FILES

/var/adm/pacct	accounting file

SEE ALSO

last(1), **sigvec**(3B), **acct**(4), **core**(4)

NAME

ld – link editor for object files

SYNOPSIS

ld [–**a** | –**r**] [–**b**] [–**G**] [–**i**] [–**m**] [–**s**] [–**t**] [–**V**] [–**B** dynamic | static]
[–**B** symbolic] [–**d** y | n] [–**D** *token*] [–**e** *epsym*] [–**F** *name*] [–**h** *name*]
[–**I** *name*] [–**L** *path*] [–**l** *x*] [–**M** *mapfile*] [–**o** *outfile*] [–**Q** y | n] [–**R** *path*]
[–**u** *symname*] [–**Y** P,*dirlist*] [–**z** defs | nodefs] [–**z** muldefs] [–**z** text]
filename . . .

DESCRIPTION

The **ld** command combines relocatable object files, performs relocation, and resolves external symbols. **ld** operates in two modes, static or dynamic, as governed by the –**d** option. In static mode, –**dn**, relocatable object files given as arguments are combined to produce an executable object file; if the –**r** option is specified, relocatable object files are combined to produce one relocatable object file. In dynamic mode, –**dy**, the default, relocatable object files given as arguments are combined to produce an executable object file that will be linked at execution with any shared object files given as arguments; if the –**G** option is specified, relocatable object files are combined to produce a shared object. In all cases, the output of **ld** is left in **a.out** by default.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. The library may be either a relocatable archive or a shared object. For an archive library, only those routines defining an unresolved external reference are loaded. The archive library symbol table (see **ar**(4)) is searched sequentially with as many passes as are necessary to resolve external references that can be satisfied by library members. Thus, the ordering of members in the library is functionally unimportant, unless there exist multiple library members defining the same external symbol. A shared object consists of a single entity all of whose references must be resolved within the executable being built or within other shared objects with which it is linked.

OPTIONS

–**a**	In static mode only, produce an executable object file; give errors for undefined references. This is the default behavior for static mode. –**a** may not be used with the –**r** option.
–**r**	Combine relocatable object files to produce one relocatable object file. **ld** will not complain about unresolved references. This option cannot be used in dynamic mode or with –**a**.
–**b**	In dynamic mode only, when creating an executable, do not do special processing for relocations that reference symbols in shared objects. Without the –**b** option, the link editor creates special position-independent relocations for references to functions defined in shared objects and arranges for data objects defined in shared objects to be copied into the memory image of the executable by the runtime linker. With the –**b** option, the output code may be more efficient, but it will be less sharable.
–**G**	In dynamic mode only, produce a shared object. Undefined symbols are allowed.

–i	Ignore **LD_LIBRARY_PATH** setting. This option is useful when an **LD_LIBRARY_PATH** setting is in effect to influence the runtime library search, which would interfere with the link editing being performed.
–m	Produce a memory map or listing of the input / output sections, together with any non-fatal multiply defined symbols, on the standard output.
–s	Strip symbolic information from the output file. Any debugging information, that is *.debug*, *.line*, and *.stab* sections, and their associated relocation entries will be removed. Except for relocatable files or shared objects, the symbol table and string table sections will also be removed from the output object file.
–t	Turn off the warning about multiply defined symbols that are not the same size.
–V	Output a message giving information about the version of **ld** being used.
–B dynamic | static	Options governing library inclusion. –**Bdynamic** is valid in dynamic mode only. These options may be specified any number of times on the command line as toggles: if the –**Bstatic** option is given, no shared objects will be accepted until –**Bdynamic** is seen. See also the –**l** option.
–B symbolic	In dynamic mode only, when building a shared object, bind references to global symbols to their definitions within the object, if definitions are available. Normally, references to global symbols within shared objects are not bound until runtime, even if definitions are available, so that definitions of the same symbol in an executable or other shared objects can override the object's own definition. **ld** will issue warnings for undefined symbols unless –**z defs** overrides.
–d y | n	When –**dy**, the default, is specified, **ld** uses dynamic linking; when –**dn** is specified, **ld** uses static linking. Also see –**B dynamic | static**.
–D *token,token,. .*	Print debugging information, as specified by each *token*, to the standard error. The special token *help* indicates the full list of tokens available.
–e *epsym*	Set the entry point address for the output file to be that of the symbol *epsym*.
–F *name*	Useful only when building a shared object. Specifies that the symbol table of the shared object is used as a "filter" on the symbol table of the shared object specified by *name.*
–h *name*	In dynamic mode only, when building a shared object, record *name* in the object's dynamic section. *name* will be recorded in executables that are linked with this object rather than the object's UNIX System file name. Accordingly, *name* will be used by the runtime linker as the name of the shared object to search for at runtime.

–I *name*	When building an executable, use *name* as the path name of the interpreter to be written into the program header. The default in static mode is no interpreter; in dynamic mode, the default is the name of the runtime linker, **/usr/lib/ld.so.1**. Either case may be overridden by –**I***name*. **exec** will load this interpreter when it loads the **a.out** and will pass control to the interpreter rather than to the **a.out** directly.
–L *path*	Add *path* to the library search directories. **ld** searches for libraries first in any directories specified by the –**L** options, and then in the standard directories. This option is useful only if it precedes the –**l** options to which it applies on the command line. The environment variable **LD_LIBRARY_PATH** may be used to supplement the library search path (see **LD_LIBRARY_PATH** below).
–l *x*	Search a library **lib***x***.so** or **lib***x***.a**, the conventional names for shared object and archive libraries, respectively. In dynamic mode, unless the –**Bstatic** option is in effect, **ld** searches each directory specified in the library search path for a file **lib***x***.so** or **lib***x***.a**. The directory search stops at the first directory containing either. **ld** chooses the file ending in **.so** if –**l***x* expands to two files whose names are of the form **lib***x***.so** and **lib***x***.a**. If no **lib***x***.so** is found, then **ld** accepts **lib***x***.a**. In static mode, or when the –**Bstatic** option is in effect, **ld** selects only the file ending in **.a**. A library is searched when its name is encountered, so the placement of –**l** is significant.
–M *mapfile*	Read *mapfile* as a text file of directives to **ld**. See *Linker and Libraries Guide* for description of mapfiles.
–o *outfile*	Produce an output object file named *outfile*. The name of the default object file is **a.out**.
–Q y | **n**	Under –**Qy**, an **ident** string is added to the **.comment** section of the output file to identify the version of the link editor used to create the file. This will result in multiple **ld idents** when there have been multiple linking steps, such as when using **ld –r**. This is identical with the default action of the **cc** command. –**Qn** suppresses version identification.
–R *path*	A colon-separated list of directories used to specify library search directories to the runtime linker. If present and not null, it is recorded in the output object file and passed to the runtime linker. Multiple instances of this option are concatenated together with each *path* separated by a colon.
–u *symname*	Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from an archive library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the command line is significant; it must be placed before the library that will define the symbol.
–Y P,*dirlist*	Change the default directories used for finding libraries. *dirlist* is a colon-separated path list.

−z defs Force a fatal error if any undefined symbols remain at the end of the link. This is the default when building an executable. It is also useful when building a shared object to assure that the object is self-contained, that is, that all its symbolic references are resolved internally.

−z nodefs Allow undefined symbols. This is the default when building a shared object. When used with executables, the behavior of references to such "undefined symbols" is unspecified.

−z muldefs Allows multiple symbol definitions. By default, multiple symbol definitions occurring between relocatable objects will result in a fatal error condition. This option suppresses the error condition, and allows the first symbol definition to be taken.

−z text In dynamic mode only, force a fatal error if any relocations against non-writable, allocatable sections remain.

ENVIRONMENT

LD_LIBRARY_PATH

A list of directories in which to search for libraries specified with the −**l** option. Multiple directories are separated by a colon. In the most general case, it will contain two directory lists separated by a semicolon:

dirlist1;*dirlist2*

If **ld** is called with any number of occurrences of −**L**, as in:

ld . . . −**L***path1* . . . −**L***pathn* . . .

then the search path ordering is:

dirlist1 path1 . . . *pathn dirlist2* **LIBPATH**

When the list of directories does not contain a semicolon, it is interpreted as *dirlist2*.

LD_LIBRARY_PATH is also used to specify library search directories to the runtime linker. That is, if **LD_LIBRARY_PATH** exists in the environment, the runtime linker will search the directories named in it, before its default directory, for shared objects to be linked with the program at execution.

Note: When running a set-user-ID or set-group-ID program, the runtime linker will only search for libraries in **/usr/lib** and any full pathname specified within the executable as a result of a runpath being specified when the executable was constructed. Any library dependencies specified as relative pathnames will be silently ignored.

LD_OPTIONS

A default set of options to **ld**. **LD_OPTIONS** is interpreted by **ld** just as though its value had been placed on the command line, immediately following the name used to invoke **ld**, as in:

ld $LD_OPTIONS . . . *other-arguments* . . .

LD_PRELOAD

A list of shared objects that are to be interpreted by the runtime linker. The specified shared objects are linked in after the program being executed and before any other shared objects that the program references.

Note: When running a set-user-ID or set-group-ID program, this option is silently ignored.

LD_RUN_PATH

An alternative mechanism for specifying a runpath to the link editor (see **-R** option). If both **LD_RUN_PATH** and the **-R** option are specified, **-R** supersedes.

LD_DEBUG

Provide a list of tokens that will cause the runtime linker to print debugging information to the standard error. The special token *help* indicates the full list of tokens available.

Note: Environment variable-names beginning with the characters '**LD_**' are reserved for possible future enhancements to **ld**.

NOTES

Options No Longer Supported

The following SunOS 4.*x*.*y* options do not have any replacement in this release: −**Bnosymbolic** (this is now the default if −**Bsymbolic** is not used), −**d**, −**dc**, and −**dp**, (these are now the default, see −**b** above to override the default), −**M**, −**S**, −**t**, −**x**, −**X**, and −**y***sym*.

The following SunOS 4.*x*.*y* options are not supported: −**align** *datum*, −**A** *name*, −**D** , −**p**, −**T**[**text**] *hex*, −**T** **data***hex*. Much of the functionality of these options can be achieved using the −**M***mapfile* option.

Obsolete Options

The following SunOS 4.*x*.*y* options are obsolete in this release: −**n**, −**N** and −**z**.

FILES

lib*x***.so**	libraries
lib*x***.a**	libraries
a.out	output file
LIBPATH	usually **/usr/ccs/lib:/usr/lib**

SEE ALSO

as(1), **cc**(1B), **ld**(1B), **exec**(2), **exit**(2), **elf**(3E), **end**(3C), **exit**(3C), **a.out**(4), **ar**(4)

Linker and Libraries Guide
Solaris Binary Compatibility Guide

NAME

ld – link editor, dynamic link editor

SYNOPSIS

/usr/ucb/ld [*options*]

AVAILABILITY

SUNWscpu

DESCRIPTION

/usr/ucb/ld is the link editor for the BSD Compatibility Package. **/usr/ucb/ld** is identical to **/usr/bin/ld** (see **ld**(1)) except that BSD libraries and routines are included *before* the base libraries and routines.

OPTIONS

/usr/ucb/ld accepts the same options as **/usr/bin/ld**, with the following exceptions:

–L*dir*	Add *dir* to the list of directories searched for libraries by **/usr/bin/ld**. Directories specified with this option are searched before **/usr/ucblib** and **/usr/lib**.
–Y LU,*dir*	Change the default directory used for finding libraries. Warning: This option may have unexpected results, and should not be used.

FILES

/usr/lib
/usr/lib/libx.a
/usr/ucblib
/usr/ucblib/libx.a

SEE ALSO

ar(1), **as**(1), **cc**(1B), **ld**(1), **lorder**(1), **strip**(1), **tsort**(1)

NAME

ldd – list dynamic dependencies of executable files or shared objects

SYNOPSIS

ldd [–**d** | –**r**] [–**s**] *filename*. . .

DESCRIPTION

ldd lists the dynamic dependencies of executable files or shared objects. If *filename* is an executable file, **ldd** lists the pathnames of all shared objects that would be loaded as a result of executing *filename*.

If *filename* is a shared object, **ldd** lists the pathnames of all shared objects that would be loaded as a result of loading *filename*. **ldd** requires shared objects to have execute permission.

ldd processes its input one file at a time. For each input file **ldd** does one of the following:

Lists the object dependencies if they exist.
Succeeds quietly if dependencies do not exit.
Prints an error message if processing fails.

OPTIONS

ldd can also check the compatibility of *filename* with the shared objects it uses. The following options indicate to **ldd** to print warnings for any unresolved symbol references that would occur if *filename* were executed.

–**d**	Check references to data objects.
–**r**	Check references to both data objects and functions.

Only one of the above options may be given during any single invocation of **ldd**.

–**s**	Displays the search path used to locate shared object dependencies.

FILES

/usr/lib/lddstub	Fake executable loaded to check the dependencies of shared objects.

SEE ALSO

ld(1), **dlopen**(3X)

Linker and Libraries Guide

DIAGNOSTICS

ldd prints the record of shared object path names to **stdout**. The optional list of symbol resolution problems are printed to **stderr**. If *filename* is not an executable file or a shared object, or if it cannot be opened for reading, a non-zero exit status is returned.

NOTES

ldd does not list shared objects explicitly attached using **dlopen**(3X).

Using the –**d** or –**r** option with shared objects can give misleading results. **ldd** does a ‘‘worst case’’ analysis of the shared objects. However, in practice some or all of the symbols reported as unresolved can be resolved by the executable file referencing the shared object.

ldd uses the same algorithm as the dynamic linker to locate shared objects.

NAME

let – shell built-in function to evaluate one or more arithmetic expressions

SYNOPSIS

ksh

let *arg* **. . .**

DESCRIPTION

ksh

Each *arg* is a separate "arithmetic expression" to be evaluated.

The exit status is **0** if the value of the last expression is non-zero, and **1** otherwise.

SEE ALSO

ksh(1), **set**(1), **typeset**(1)

NAME

lex – lexical analysis program generator

SYNOPSIS

lex [**–cntv**] [**–e** | **–w**] [**–V** **–Q** [**y** | **n**]][*filename*] . . .

DESCRIPTION

lex generates programs to be used in simple lexical analysis of text. Each *filename* (the standard input by default) contains regular expressions to search for, and actions written in C to be executed when expressions are found.

A C source program, **lex.yy.c** is generated, to be compiled as follows:

```
cc lex.yy.c –ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized. The actual string matched is left in **yytext**, an external character array (a **wchar_t** array when the –**w** option is given).

Matching is done in order of the strings in the file. The strings may contain square braces to indicate character classes, as in **[abx–z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators ∗, **+** and **?**, which mean, respectively, any nonnegative number, any positive number, or either zero or one occurrences of the previous character or character-class. The “dot” character (‘**.**’) is the class of all characters except NEWLINE.

Parentheses for grouping and vertical bar for alternation are also supported. The notation *r*{*d*,*e*} in a rule indicates instances of regular expression *r* between *d* and *e*. It has a higher precedence than **|**, but lower than that of ∗, **?**, **+**, or concatenation. The **^** (carat character) at the beginning of an expression permits a successful match only immediately after a NEWLINE, and the **$** character at the end of an expression requires a trailing NEWLINE.

The **/** character in an expression indicates trailing context; only the part of the expression up to the slash is returned in **yytext**, although the remainder of the expression must follow in the input stream.

An operator character may be used as an ordinary symbol if it is within ´’ symbols or is preceded by ‘\’.

Three subroutines defined as macros are expected: **input( )** to read a character; **unput(***c***)** to replace a character read; and **output(***c***)** to place an output character. They are defined in terms of the standard streams, but you can override them. For C++ code, **input( )** is renamed **lex_input( )**, and **ouput( )** is renamed **lex_output( )** to avoid name conflicts with iostreams. The program generated is named **yylex( )**, and the lex library **libl.a** contains a **main( )** which calls it. The action REJECT on the right side of the rule rejects this match and executes the next suitable match; the function **yymore( )** accumulates additional characters into the same **yytext**; and the function **yyless(***p***)** pushes back the portion of the string matched beginning at *p*, which should be between **yytext** and **yytext**+**yyleng**. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

In a **lex** program, any line beginning with a blank is assumed to contain only C text and is copied; if it precedes **%%** it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a **%%**, as in YACC. Lines preceding **%%** which begin with a non-blank character define the string on the left to be the remainder of the line; it can be used later by surrounding it with **{}**. Note: curly brackets do not imply parentheses; only string substitution is done.

The external names generated by **lex** all begin with the prefix **yy** or **YY**.

Certain table sizes for the resulting finite-state machine can be set in the definitions section:

%p *n*	number of positions is *n* (default 2000)
%n *n*	number of states is *n* (default 500)
%e *n*	number of parse tree nodes is *n* (default 1000)
%a *n*	number of transitions is *n* (default 3000)

The use of one or more of the above automatically implies the **–v** option, unless the **–n** option is used.

Programs generated by **lex**(1) need either the **–e** or **–w** option to handle input that contains EUC characters from supplementary codesets. If neither of these options is specified, **yytext** is of the type **char[ ]**, and the generated program can handle only ASCII characters.

When the **–e** option is used, **yytext** is of the type **unsigned char[ ]** and **yyleng** gives the total number of *bytes* in the matched string. With this option, the macros **input()**, **unput(***c***)**, and **output(***c***)** should do a byte-based I/O in the same way as with the regular ASCII **lex**(1). Two more variables are available with the **–e** option, **yywtext** and **yywleng**, which behave the same as **yytext** and **yyleng** would under the **–w** option.

When the **–w** option is used, **yytext** is of the type **wchar_t[ ]** and **yyleng** gives the total number of *characters* in the matched string. If you supply your own **input()**, **unput(***c***)**, or **output(***c***)** macros with this option, they must return or accept EUC characters in the form of wide character (*wchar_t*). This allows a different interface between your program and the lex internals, to expedite some programs.

When either the **–e** or **–w** option is used, the generated C program must be linked with the wide character library **libw.a** using the **–lw** linker flag.

Pattern Matching

When either the **–e** or **–w** option is used, patterns used in rules can include characters from both primary and supplementary codesets. The generated program performs pattern matching correctly on an input stream containing EUC characters from supplementary codesets.

You may use any valid EUC characters in a character range **[***A*–*Z***]** as long as *A* and *Z* belong to the same codeset.

"." matches any character from any codeset (except NEWLINE).

International Caveats

Start condition names must consist solely of ASCII characters.

The "%T" directive can not be used when either the −**w** or −**e** option is used.

The default main() found in the lex library (libl.a) does not have a **setlocale**(3C) call. Thus, the resulting program would not recognize non-ASCII characters correctly. You have to supply your own main( ) in order to have your program handle EUC characters correctly. The simplest main( ) would be:

```
#include <locale.h>
main( ){
        setlocale(LC_ALL, "");
        yylex( );
}
```

OPTIONS

−**c**	Indicates C actions and is the default.
−**e**	Generate a program that can handle EUC characters (cannot be used with the −**w** option). **yytext[ ]** is of type **unsigned**char[ ]**.**
−**n**	Opposite of −**v**; −**n** is the default.
−**t**	Place the result on the standard output instead of in file **lex.yy.c**.
−**v**	Print a one-line summary of statistics of the generated analyzer.
−**w**	Generate a program that can handle EUC characters (cannot be used with the −**e** option). Unlike the −**e** option, **yytext[ ]** is of type **wchar_t[ ]**.
−**V**	Print out version information on standard error.
−**Q[y | n]**	Print out version information to output file **lex.yy.c** by using −**Qy .** The −**Qn** option does not print out version information and is the default.

EXAMPLES

The command line,

```
lex lexcommands
```

draws **lex** instructions from the file **lexcommands**, and places the output in **lex.yy.c**.

The following example **lex** program converts uppercase to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A−Z]    putchar (yytext[0]+´a´−´A´);
[ ]+$    ;
[ ]+     putchar(´ ´);
```

INTERNATIONAL EXAMPLES

The following is a similar program for the "**japanese**" locale environment:

```
%%
[\x30001221-\x30001273]   putwchar (yytext[0]+0x0080);
```

```
[ \x300010a1]+$            ;
[ \x300010a1]+             putchar(´ ´);
%%
#include <locale.h>
main( ){
        setlocale(LC_ALL, "");
        yylex( );
}
```

This program converts every hiragana character (of which the EUC wide character value is between 0x30001221 and 0x30001273) to the corresponding katakana character. It also recognizes double-space character (0x300010a1). 0x0080 is the offset between corresponding hiragana and katakana characters when represented in wide characters. Note that use of the hexadecimal escape sequence in this example is not really needed. The corresponding EUC characters could have been used instead.

This program must be compiled with the **–lw** option and linked with the wide character library **libw.a**. Compilation and execution must be done in an environment where either the LANG or LC_CTYPE environment variable is set to **japanese**. The command line for compiling this program would be:

```
% lex –w sample.l
% cc –o sample lex.yy.c –ll –lw
```

FILES

lex.yy.c	default output file when **–t** is not specified
/usr/ccs/lib/libl.a	lex library
ncform	
nceucform	C-program prototypes

SEE ALSO

sed(1), **yacc**(1), **setlocale**(3C)

lex in *Programming Utilities Guide*

NOTES

Ratfor is no longer supported as a host language.

The way to use hexadecimal escape sequences for multibyte characters differs from the versions of lex of previous release of SunOS Asian Language Environment, namely JLE, KLE, CLE and HLE. In these versions, a multibyte character was written as a sequence of hexadecimal escape sequences, one per byte, rather than as one hexadecimal escape sequence representing the character's wide character value.

NAME

limit, ulimit, unlimit – shell built-in functions to set / get limitations on the system resources available to the current shell and it’s descendents.

SYNOPSIS

sh

ulimit [–[**HS**] [**a**| **cdfnstv**]]
ulimit [–[**HS**]] **c**| **d**| **f**| **n**| **s**| **t**| **v**]] *limit*

csh

limit [–**h**] [*resource* [*max-use*]]

unlimit [–**h**] [*resource*]

ksh

ulimit [–**HSacdfmnpstv**] [*limit*]

DESCRIPTION

sh

ulimit prints or sets hard or soft resource limits. These limits are described in **getrlimit**(2).

If *limit* is not present, **ulimit** prints the specified limits. Any number of limits may be printed at one time. The –**a** option prints all limits.

If *limit* is present, **ulimit** sets the specified limit to *limit*. The string **unlimited** requests the largest valid limit. Limits may be set for only one resource at a time. Any user may set a soft limit to any value below the hard limit. Any user may lower a hard limit. Only a super-user may raise a hard limit; see **su**(1M).

The –**H** option specifies a hard limit. The –**S** option specifies a soft limit. If neither option is specified, **ulimit** will set both limits and print the soft limit.

The following options specify the resource whose limits are to be printed or set. If no option is specified, the file size limit is printed or set.

–**c**	maximum core file size (in 512-byte blocks)
–**d**	maximum size of data segment or heap (in kbytes)
–**f**	maximum file size (in 512-byte blocks)
–**n**	maximum file descriptor plus 1
–**s**	maximum size of stack segment (in kbytes)
–**t**	maximum CPU time (in seconds)
–**v**	maximum size of virtual memory (in kbytes)

csh

limit limits the consumption by the current process or any process it spawns, each not to exceed *max-use* on the specified *resource*. If *max-use* is omitted, print the current limit; if *resource* is omitted, display all limits.

–**h**	Use hard limits instead of the current limits. Hard limits impose a ceiling on the values of the current limits. Only the privileged user may raise the hard limits.

resource is one of:

cputime	Maximum CPU seconds per process.
filesize	Largest single file allowed.

datasize	Maximum data size (including stack) for the process.
stacksize	Maximum stack size for the process.
coredumpsize	Maximum size of a core dump (file).
descriptors	Maximum number of file descriptors.
memorysize	Maximum size of virtual memory.

max-use is a number, with an optional scaling factor, as follows:

*n***h**	Hours (for **cputime**).
*n***k**	*n* kilobytes. This is the default for all but **cputime**.
*n***m**	*n* megabytes or minutes (for **cputime**).
*mm***:***ss*	Minutes and seconds (for **cputime**).

unlimit removes a limitation on *resource*. If no *resource* is specified, then all resource limitations are removed. See the description of the **limit** command for the list of resource names.

–h Remove corresponding hard limits. Only the privileged user may do this.

ksh

ulimit sets or displays a resource limit. The available resources limits are listed below. Many systems do not contain one or more of these limits. The limit for a specified resource is set when *limit* is specified. The value of *limit* can be a number in the unit specified below with each resource, or the value **unlimited**. The **H** and **S** flags specify whether the hard limit or the soft limit for the given resource is set. A hard limit cannot be increased once it is set. A soft limit can be increased up to the value of the hard limit. If neither the **H** or **S** options is specified, the limit applies to both. The current resource limit is printed when *limit* is omitted. In this case the soft limit is printed unless **H** is specified. When more that one resource is specified, then the limit name and unit is printed before the value.

–a	Lists all of the current resource limits.
–c	The number of 512-byte blocks on the size of core dumps.
–d	The number of K-bytes on the size of the data area.
–f	The number of 512-byte blocks on files written by child processes (files of any size may be read).
–m	The number of K-bytes on the size of physical memory.
–n	The number of file descriptors plus 1.
–p	The number of 512-byte blocks for pipe buffering.
–s	The number of K-bytes on the size of the stack area.
–t	The number of seconds to be used by each process.
–v	The number of K-bytes for virtual memory.

If no option is given, **–f** is assumed.

SEE ALSO

csh(1), **ksh**(1), **sh**(1), **su**(1), **getrlimit**(2)

NAME

line – read one line

SYNOPSIS

line

AVAILABILITY

SUNWcsu

DESCRIPTION

line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1), **read**(2)

NAME

listusers – list user login information

SYNOPSIS

listusers [–**g** *groups*] [–**l** *logins*]

AVAILABILITY

SUNWcsu

DESCRIPTION

Executed without any options, this command lists all user logins sorted by login. The output shows the login ID and the account field value from the system's password database as specified by **/etc/nsswitch.conf**.

OPTIONS

–**g** *groups*	Lists all user logins belonging to **group**, sorted by login. Multiple groups can be specified as a comma-separated list.
–**l** *logins*	Lists the user login or logins specified by **logins**, sorted by login. Multiple logins can be specified as a comma-separated list.

SEE ALSO

nsswitch.conf(4)

NOTES

A user login is one that has a UID of 100 or greater.

The –**l** and –**g** options can be combined. User logins will only be listed once, even if they belong to more than one of the selected groups.

NAME

ln – make hard or symbolic links to files

SYNOPSIS

ln [–**f**] [–**n**] [–**s**] *filename1* [*filename2*. . .*filenamen*] *target*

AVAILABILITY

SUNWcsu

DESCRIPTION

The **ln** command links *filenamen* to *target* by creating a directory entry that refers to *target*. By using **ln** with one or more file names, the user may create one or more links to *target*.

The **ln** command may be used to create both hard links and symbolic links; by default it creates hard links. A hard link to a file is indistinguishable from the original directory entry. Any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

Without the –**s** option, **ln** is used to create hard links. *filenamen* is linked to *target*. If *target* is a directory, another file named *filenamen* is created in *target* and linked to the original *filenamen*. If *target* is a file, its contents are overwritten.

If **ln** determines that the mode of *target* forbids writing, it will print the mode (see **chmod**(2)), ask for a response, and read the standard input for one line. If the line begins with **y**, the link occurs, if permissible; otherwise, the command exits.

OPTIONS

–**f** **ln** will link files without questioning the user, even if the mode of *target* forbids writing. Note: This is the default if the standard input is not a terminal.

–**n** If the linkname is an existing file, do not overwrite the contents of the file. The –**f** option overrides this option.

–**s** **ln** will create a symbolic link. A symbolic link contains the name of the file to which it is linked. Symbolic links may span file systems and may refer to directories.

If the –**s** option is used with two arguments, *target* may be an existing directory or a non-existent file. If *target* already exists and is not a directory, an error is returned. *filenamen* may be any path name and need not exist. If it exists, it may be a file or directory and may reside on a different file system from *target*. If *target* is an existing directory, a file is created in directory *target* whose name is *filenamen* or the last component of *filenamen*. This file is a symbolic link that references *filenamen*. If *target* does not exist, a file with name *target* is created and it is a symbolic link that references *filenamen*.

If the –**s** option is used with more than two arguments, *target* must be an existing directory or an error will be returned. For each *filenamen*, a file is created in *target* whose name is *filenamen* or its last component; each new *filenamen* is a symbolic link to the original *filenamen*. The *files* and *target* may reside on different file systems.

File permissions for *target* may be different than those displayed with a –**l** listing of the **ls**(1) command. To display the permissions of *target* use **ls** –**lL**. See **stat**(2) for more information.

SEE ALSO **chmod**(1), **cp**(1), **ls**(1), **mv**(1), **rm**(1), **link**(1M), **readlink**(2), **stat**(2), **symlink**(2)

NAME

ln – make hard or symbolic links to files

SYNOPSIS

/usr/ucb/ln [**–fs**] *filename* [*linkname*]
/usr/ucb/ln [**–fs**] *pathname*. . . *directory*

AVAILABILITY

SUNWscpu

DESCRIPTION

/usr/ucb/ln creates an additional directory entry, called a link, to a file or directory. Any number of links can be assigned to a file. The number of links does not affect other file attributes such as size, protections, data, etc.

filename is the name of the original file or directory. *linkname* is the new name to associate with the file or filename. If *linkname* is omitted, the last component of *filename* is used as the name of the link.

If the last argument is the name of a directory, symbolic links are made in that directory for each *pathname* argument; **/usr/ucb/ln** uses the last component of each *pathname* as the name of each link in the named *directory*.

A hard link (the default) is a standard directory entry just like the one made when the file was created. Hard links can only be made to existing files. Hard links cannot be made across file systems (disk partitions, mounted file systems). To remove a file, all hard links to it must be removed, including the name by which it was first created; removing the last hard link releases the inode associated with the file.

A symbolic link, made with the **–s** option, is a special directory entry that points to another named file. Symbolic links can span file systems and point to directories. In fact, you can create a symbolic link that points to a file that is currently absent from the file system; removing the file that it points to does not affect or alter the symbolic link itself.

A symbolic link to a directory behaves differently than you might expect in certain cases. While an **ls**(1) on such a link displays the files in the pointed-to directory, an '**ls –l**' displays information about the link itself:

```
example% /usr/ucb/ln –s dir link
example% ls link
file1 file2 file3 file4
example% ls –l link
lrwxrwxrwx  1 user          7 Jan 11 23:27 link → dir
```

When you **cd**(1) to a directory through a symbolic link, you wind up in the pointed-to location within the file system. This means that the parent of the new working directory is not the parent of the symbolic link, but rather, the parent of the pointed-to directory. For instance, in the following case the final working directory is **/usr** and not **/home/user/linktest**.

```
example% pwd
/home/user/linktest
example% /usr/ucb/ln –s /var/tmp symlink
example% cd symlink
```

```
example% cd ..
example% pwd
/usr
```

C shell user's can avoid any resulting navigation problems by using the **pushd** and **popd** built-in commands instead of **cd**.

OPTIONS

–f Force a hard link to a directory. This option is only available to the super-user, and should be used with extreme caution.

–s Create a symbolic link or links.

EXAMPLES

The commands below illustrate the effects of the different forms of the **/usr/ucb/ln** command:

```
example% /usr/ucb/ln file link
example% ls –F file link
file   link
example% /usr/ucb/ln –s file symlink
example% ls –F file symlink
file   symlink@
example% ls –li file link symlink
 10606 -rw-r--r--  2 user          0 Jan 12 00:06 file
 10606 -rw-r--r--  2 user          0 Jan 12 00:06 link
 10607 lrwxrwxrwx  1 user          4 Jan 12 00:06 symlink → file
example% /usr/ucb/ln –s nonesuch devoid
example% ls –F devoid
devoid@
example% cat devoid
devoid: No such file or directory
example% /usr/ucb/ln –s /proto/bin/* /tmp/bin
example% ls –F /proto/bin /tmp/bin
/proto/bin:
x*     y*     z*

/tmp/bin:
x@     y@     z@
```

SEE ALSO

cp(1), **ls**(1), **mv**(1), **rm**(1), **link**(2), **readlink**(2), **stat**(2), **symlink**(2)

NOTES

When the last argument is a directory, simple basenames should not be used for *pathname* arguments. If a basename is used, the resulting symbolic link points to itself:

```
example% /usr/ucb/ln –s file /tmp
example% ls –l /tmp/file
lrwxrwxrwx  1 user          4 Jan 12 00:16 /tmp/file → file
example% cat /tmp/file
/tmp/file: Too many levels of symbolic links
```

To avoid this problem, use full pathnames, or prepend a reference to the **PWD** variable to files in the working directory:

```
example% rm /tmp/file
example% /usr/ucb/ln –s $PWD/file /tmp
lrwxrwxrwx  1 user 4      Jan 12 00:16 /tmp/file → /home/user/subdir/file
```

NAME

loadfont – display or change font information in the RAM of the video card on an x86 system in text mode

SYNOPSIS

loadfont [–**f** *BDF_file* | –**c** *codeset*] [–**m** *mode*] [–**d**]

AVAILABILITY

x86

SUNWcsu

DESCRIPTION

The **loadfont** utility allows a user to load and activate a different font into the RAM of the video card used by the console of the Solaris for x86 operating system in text mode. It can also be used to display information about the fonts currently in use. In addition, the –**m** option can be used to change the size of the characters on the screen; it can also be used to change the number of lines per screen. **loadfont** will always read from standard output; this will allow a system administrator to use it from a remote terminal.

When used without arguments, **loadfont** displays the different ways the command can be used, as shown in the synopsis.

Options

–**f** *BDF_file*

This command reads the contents of *BDF_file* and subsequently loads the font specified in the file into the RAM of the video card. The file must be in the Binary Distribution Format version 2.1 as developed by Adobe Systems, Inc. (See **loadfont**(4).)

–**c** *codeset*

codeset is the name of a codeset available for the current font size. This font will be loaded into the RAM of the video card and activated. Use **?** to find out the valid *codesets* available. This option is a shorthand form of –**f**.

–**m** *mode*

This option will attempt to change the mode of the console as specified. This will result in having a different font size and / or different number of lines and columns on the screen. Use **?** to find out the valid *modes* available.

–**d**

This *read*s the font information from the video RAM and *write*s it to standard output in a format compatible with the Binary Distribution Format version 2.1 as developed by Adobe Systems, Inc. (See **loadfont**(4).)

Fonts

A font is the representation of characters by images. The need to use different fonts can be imposed by:

1. The codeset used to represent the characters internally.
2. The resolution used to display the characters.

Each font contains exactly 256 images. All supported fonts are fixed size (constant width and constant height), i.e., each character takes the same amount of space on the screen. When the monitor is not being used in graphics mode, the **loadfont** utility allows a user to modify the font used by the video card, so different images are displayed on the screen of the console for the various characters. The same video card may support different text modes. Video cards typically differ by the number of pixels they use to represent a single character. On any given video card, the same number of pixels is used for each character. For the standard VGA video cards, 8 by 16 (8 horizontally and 16 vertically) resolution is supported:

When **loadfont** is invoked to modify the existing font, it will attempt to do so for the font size currently in use. Use the **–m** option to switch to another font size.

loadfont and pcmapkeys

There is an almost one-to-one relationship between the use of the **loadfont** utility and the **pcmapkeys** utility. Whereas **loadfont** is used to list or modify the images that correspond with the various characters, the **pcmapkeys** utility is used to determine how characters are generated from the keyboard and which code (a single byte code) will be used to represent the character internally. The default representation is the ISO 8859-1 codeset.

When a different codeset is used, both a different **pcmapkeys** input file and a different font set are required. If the default font does not satisfy your needs (because a different font size or a customized font is required, e.g., a Greek font), a **loadfont** description file to be used with the **–f** option is needed. A sample file that describes the IBM extended ASCII font for an 8 by 16 resolution is supplied (**437.bdf**). A second sample file, **646g.bdf**, contains a font file for German ASCII. See **pcmapkeys**(1) and **loadfont**(4) for additional details.

WARNING

When an attempt is made to switch to a mode that the video card does not support, you will get a blank screen. There is nothing wrong with the system; as super-user, simply type in the command to set the mode back, e.g.:

```
loadfont –m V80x25
```

FILES

/usr/share/lib/fonts/8859-1.bdf	the Binary Distribution Format (BDF) file for the default fonts
/usr/share/lib/fonts/437.bdf	sample Binary Distribution Format (BDF) file for IBM 437 font on a VGA
/usr/share/lib/fonts/646g.bdf	sample BDF file for German ASCII

SEE ALSO

pcmapkeys(1), **loadfont**(4)

NOTE

The default fonts on the system are those of the ISO 8859-1 codeset. The optional IBM DOS 437 codeset is supported *only* at internationalization level 1. That is, if you choose to download fonts of the optional IBM DOS 437 codeset, there will be no support for non-standard U.S. date, time, currency, numbers, unit, and collation. There will be no support for non-English message and text presentation, and no multi-byte character support. Therefore, non-Windows users should only use IBM DOS 437 codeset in the default C locale.

NAME

loadkeys, dumpkeys – load and dump keyboard translation tables

SYNOPSIS

loadkeys [–**e**] [*filename*]

dumpkeys

AVAILABILITY

SPARC

SUNWloc

DESCRIPTION

loadkeys reads the file specified by *filename*, and modifies the keyboard streams module's translation tables. If no file is specified, and the keyboard is a Type-4 keyboard, a default file for the layout indicated by the DIP switches on the keyboard. The file is in the format specified by **keytables**(4).

If the layout code in the DIP switches on the keyboard has the hexadecimal value **0x***dd*, the file loaded by **loadkeys** by default is **/usr/share/lib/keytables/layout_***dd*. These files specify only the entries that change between the different Type-4 keyboard layouts.

dumpkeys writes, to the standard output, the current contents of the keyboard streams module's translation tables, in the format specified by **keytables**(4).

OPTIONS

–**e** **loadkeys** loads the requested keytable layout information into the EEPROM keytables. This allows the console monitor to select the correct key-values when a non-US keyboard is connected to the workstation.

FILES

/usr/share/lib/keytables/layout_*dd*	default keytable files

SEE ALSO

kbd(1), **keytables**(4), **kb**(7)

BUGS

The –**e** option requires EEPROM rev 2.6 or later.

NAME

logger – add entries to the system log

SYNOPSIS

/usr/ucb/logger [**–i**] [**–f** *filename*] [**–p** *priority*] [**–t** *tag*] [*message*] . . .

DESCRIPTION

logger provides a method for adding one-line entries to the system log file from the command line. One or more *message* arguments can be given on the command line, in which case each is logged immediately. If this is unspecified, either the file indicated with **–f** or the standard input is added to the log. Otherwise, a *filename* can be specified, in which case each line in the file is logged. If neither is specified, **logger** reads and logs messages on a line-by-line basis from the standard input.

OPTIONS

–i	Log the process ID of the **logger** process with each line.
–f *filename*	Use the contents of *filename* as the message to log.
–p *priority*	Enter the message with the specified *priority*. The message priority can be specified numerically, or as a *facility***.***level* pair. For example, '**–p local3.info**' assigns the message priority to the **info** level in the **local3** facility. The default priority is **user.notice**.
–t *tag*	Mark each line added to the log with the specified *tag*.

EXAMPLES

The following example:

```
example% logger System rebooted
```

logs the message '**System rebooted**' to the default priority level **notice** to be treated by **syslogd** as are other messages to the facility **user**.

The next example:

```
logger –p local0.notice –t HOSTIDM –f /dev/idmc
```

reads from the file **/dev/idmc** and logs each line in that file as a message with the tag '**HOSTIDM**' at priority level **notice** to be treated by **syslogd** as are other messages to the facility **local0**.

SEE ALSO

syslogd(1M), **syslog**(3)

NAME

logger – add entries to the system log

SYNOPSIS

/usr/ucb/logger [**–f** *filename*] [**–i**] [**–p** *priority*] [**–t** *tag*] [*message*] . . .

AVAILABILITY

SUNWscpu

DESCRIPTION

logger provides a method for adding one-line entries to the system log file from the command line. One or more *message* arguments can be given on the command line, in which case each is logged immediately. If *message* is unspecified, either the file indicated with **–f** or the standard input is added to the log. Otherwise, a *filename* can be specified, in which case each line in the file is logged. If neither is specified, **logger** reads and logs messages on a line-by-line basis from the standard input.

OPTIONS

–i	Log the process ID of the **logger** process with each line.
–f *filename*	Use the contents of *filename* as the message to log.
–p *priority*	Enter the message with the specified *priority*. The message priority can be specified numerically, or as a *facility.level* pair. For example, **'–p local3.info'** assigns the message priority to the **info** level in the **local3** facility. The default priority is **user.notice**.
–t *tag*	Mark each line added to the log with the specified *tag*.

EXAMPLES

The command:

```
example% logger System rebooted
```

will log the message **'System rebooted'** to the facility at priority **notice** to be treated by **syslogd** as other messages to the facility **notice** are.

The next command:

```
example% logger –p local0.notice –t HOSTIDM –f /dev/idmc
```

will read from the file **/dev/idmc** and will log each line in that file as a message with the tag **'HOSTIDM'** at priority **notice** to be treated by **syslogd** as other messages to the facility **local0** are.

SEE ALSO

syslogd(1M), **syslog**(3)

NAME

login – sign on to the system

SYNOPSIS

login [**–d** *device*] [**–h** *hostname* [*terminal*] | **–r** *hostname*] [*name* [*environ* ...]]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **login** command is used at the beginning of each terminal session and allows you to identify yourself to the system. It is invoked by the system when a connection is first established and after the previous user has terminated the login shell by issuing the **exit** command.

If **login** is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

login asks for your user name (if it is not supplied as an argument), and if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

If there are no lower-case characters in the first line of input processed, **login** assumes the connecting TTY is an upper-case-only terminal and sets the port's **termio**(7) options to reflect this.

If you make any mistake in the login procedure, the message:

Login incorrect

is printed and a new login prompt will appear. If you make five incorrect login attempts, all five may be logged in **/var/adm/loginlog** (if it exists) and the TTY line will be dropped.

If you do not complete the login successfully within a certain period of time, it is likely that you will be silently disconnected.

After a successful login, accounting files are updated; device owner, group, and permissions are set according to the contents of the **/etc/logindevperm** file (see **logindevperm**(4)); and the time you last logged in is printed.

The user-ID, group-ID, supplementary group list, and working directory are initialized, and the command interpreter (usually **ksh**) is started.

The basic *environment* is initialized to:

HOME=*your-login-directory*
LOGNAME=*your-login-name*
PATH=/usr/bin:

SHELL=*last-field-of-passwd-entry*
MAIL=/var/mail/*your-login-name*
TZ=*timezone-specification*

For Bourne shell and Korn shell logins, the shell executes **/etc/profile** and **$HOME/.profile** (if it exists). For C shell logins, the shell executes **/etc/.login**, **$HOME/.cshrc**, and **$HOME/.login**. The default **/etc/profile** and **/etc/.login** files check quotas (see **quota**(1M)), print **/etc/motd** and check for mail. None of the messages are printed if the file **$HOME/.hushlogin** exists. The name of the command interpreter is set to – (dash), followed by the last component of the interpreter's path name (for example, –**sh**).

If the *login-shell* field in the password file (see **passwd**(4)) is empty, then the default command interpreter, **/usr/bin/sh**, is used. If this field is ∗ (asterisk), then the named directory becomes the root directory. At that point **login** is re-executed at the new level which must have its own root structure.

The environment may be expanded or modified by supplying additional arguments to **login**, either at execution time or when **login** requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as:

L*n*=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an **=** are placed in the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. **login** understands simple single-character quoting conventions. Typing a ‘ \ ’ (backslash) in front of a character quotes it and allows the inclusion of such characters as spaces and tabs.

To enable remote logins by root, edit the **/etc/default/login** file by inserting a ‘ # ’ (pound-sign) before the **CONSOLE=/dev/console** entry. See **FILES** below.

OPTIONS

–**d** *device* **login** accepts a device option, *device. device* is taken to be the path name of the TTY port **login** is to operate on. The use of the device option can be expected to improve **login** performance, since **login** will not need to call **ttyname**(3C). The –**d** option is available only to users whose **UID** and effective **UID** are root. Any other attempt to use –**d** will cause **login** to quietly exit.

–**h** *hostname* [*terminal*]
used by **in.telnetd**(1M) to pass information about the remote host and terminal type.

–**r** *hostname*
used by **in.rlogind**(1M) to pass information about the remote host.

FILES

$HOME/.cshrc	initial commands for each csh
$HOME/.hushlogin	suppresses login messages
$HOME/.login	user's login commands for csh

$HOME/.profile	user's login commands for sh and ksh
$HOME/.rhosts	private list of trusted hostname / username combinations
/etc/.login	system-wide csh login commands
/etc/logindevperm	login-based device permissions
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system-wide sh and ksh login commands
/etc/shadow	list of users' encrypted passwords
/usr/bin/sh	user's default command interpreter
/var/adm/lastlog	time of last login
/var/adm/loginlog	record of failed login attempts
/var/adm/utmp	accounting
/var/adm/wtmp	accounting
/var/mail/*your-name*	mailbox for user *your-name*
/etc/default/login	Default value can be set for the following flags in **/etc/default/login**. For example: TIMEZONE=EST5EDT or PASSREQ=YES.

TIMEZONE:	Sets the TZ environment variable of the shell.
HZ:	Sets the HZ environment variable of the shell.
ULIMIT:	Sets the file size limit for the login. Units are disk blocks. Default is zero (no limit).
CONSOLE:	If set, root can login on that device only. This will not prevent execution of remote commands with **rsh**(1). Comment out this line to allow login by root.
PASSREQ:	Determines if login requires a password.
ALTSHELL:	Determines if login should set the **SHELL** environment variable.
PATH:	Sets the initial shell **PATH** variable.
SUPATH:	Sets the initial shell **PATH** variable for root.
TIMEOUT:	Sets the number of seconds (between 0 and 900) to wait before abandoning a login session.
UMASK:	Sets the initial shell file creation mode mask. See **umask**(1).
SYSLOG:	determines whether the **syslog**(3) LOG_AUTH facility should be used to log all root logins at level LOG_NOTICE and multiple failed login attempts at LOG_CRIT.
SLEEPTIME	If present sets the number of seconds to wait before login failure is printed to the screen and another login and another login attempt is allowed. Default is 4 seconds; Minimum is 0 seconds. Maximum is 5 seconds.

SEE ALSO

csh(1), **ksh**(1), **mail**(1), **mailx**(1), **passwd**(1), **rlogin**(1), **rsh**(1), **sh**(1), **shell_builtins**(1), **telnet**(1), **admintool**(1M), **in.rlogind**(1M), **in.telnetd**(1M), **logins**(1M), **newgrp**(1M), **quota**(1M), **su**(1M), **syslogd**(1M), **useradd**(1M), **userdel**(1M), **syslog**(3), **hosts.equiv**(4),

logindevperm(4), **loginlog**(4), **passwd**(4), **profile**(4), **shadow**(4), **environ**(5)

DIAGNOSTICS

Login incorrect
The user name or the password cannot be matched.

Not on system console
Root login denied. Check the **CONSOLE** setting in **/etc/default/login**.

No directory! Logging in with home=/
The user's home directory named in the **passwd**(4) database cannot be found or has the wrong permissions. Contact your system administrator.

No shell
Cannot execute the shell named in the **passwd**(4) database. Contact your system administrator.

WARNINGS

If you use the **CONSOLE** setting to disable root logins, you should arrange that remote command execution by root is also disabled. See **rsh**(1), **rcmd**(3N), and **hosts.equiv**(4) for further details.

NAME

logname – get the name of the user running the process

SYNOPSIS

logname

AVAILABILITY

SUNWesu

DESCRIPTION

logname returns the name of the user running the process.

FILES

/etc/profile

SEE ALSO

env(1), **login**(1), **cuserid**(3S), **environ**(5)

EXIT CODES

logname returns the following exit codes:

0	If **logname** is successful.
1	If **logname** cannot identify the user.

NAME

logout – shell built-in function to exit from a login session

SYNOPSIS

csh

logout

DESCRIPTION

csh

Terminate a login shell.

SEE ALSO

csh(1), **login**(1)

NAME

look – find words in the system dictionary or lines in a sorted list

SYNOPSIS

/usr/bin/look [**–d**] [**–f**] [**–t***c*] *string* [*filename*]

AVAILABILITY

SUNWdoc

DESCRIPTION

The **look** command consults a sorted *filename* and prints all lines that begin with *string*.

If no *filename* is specified, **look** uses **/usr/share/lib/dict/words** with collating sequence **–df**.

look limits the length of a word to search for to 256 characters.

OPTIONS

–d	Dictionary order. Only letters, digits, TAB and SPACE characters are used in comparisons.
–f	Fold case. Upper case letters are not distinguished from lower case in comparisons.
–t*c*	Set termination character. All characters to the right of *c* in *string* are ignored.

FILES

/usr/share/lib/dict/words	spelling list

SEE ALSO

grep(1), **sort**(1)

NAME

lookbib – find references in a bibliographic database

SYNOPSIS

lookbib *database*

AVAILABILITY

SUNWdoc

DESCRIPTION

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '**%**', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '**%**'.

lookbib uses an inverted index made by **indxbib** to find sets of bibliographic references. It reads keywords typed after the '**>**' prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another '**>**' prompt.

It is possible to search multiple databases, as long as they have a common index made by **indxbib**(1). In that case, only the first argument given to **indxbib** is specified to **lookbib.**

If **lookbib** does not find the index files (the **.i[abc]** files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a **.ig** suffix, suitable for use with **fgrep** (see **grep**(1)). **lookbib** then uses this **fgrep** file to find references. This method is simpler to use, but the **.ig** file is slower to use than the **.i[abc]** files, and does not allow the use of multiple reference files.

FILES

x.**ia**	
x.**ib**	
x.**ic**	index files
x.ig	reference file

SEE ALSO

addbib(1), **grep**(1), **indxbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1)

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

lorder – find ordering relation for an object or library archive

SYNOPSIS

lorder *filename* . . .

DESCRIPTION

The input is one or more object or library archive *filenames* (see **ar**(1)). The standard output is a list of pairs of object file or archive member names; the first file of the pair refers to external identifiers defined in the second. The output may be processed by **tsort**(1) to find an ordering of a library suitable for one-pass access by **ld**. Note: The link editor **ld** is capable of multiple passes over an archive in the portable archive format (see **ar**(4)) and does not require that **lorder** be used when building an archive. The usage of the **lorder** command may, however, allow for a more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar –cr library `lorder *.o | tsort`
```

FILES

TMPDIR / ∗**symref**	temporary files
TMPDIR / ∗**symdef**	temporary files
TMPDIR	usually **/var/tmp** but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**() in **tmpnam**(3S))

SEE ALSO

ar(1), **ld**(1), **tsort**(1), **tmpnam**(3S), **ar**(4)

NOTES

lorder will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be **.o**.

The length of the filename for **TMPDIR** is limited to whatever **sed** allows.

NAME

lp, cancel – send / cancel requests to an LP print service

SYNOPSIS

lp [–**c**] [–**m**] [–**p**] [–**s**] [–**w**] [–**d** *dest*] [–**f** *form-name* [–**d** **any**]]
[–**H** *special-handling*] [–**n** *number*] [–**o** *option*] [–**P** *page-list*] [–**q** *priority-level*]
[–**S** *character-set* [–**d** **any**]] [–**S** *print-wheel* [–**d** **any**]] [–**t** *title*]
[–**T** *content-type* [–**r**]] [–**y** *mode-list*] [*file* . . .]

lp –**i** *request-ID...* [–**c**] [–**m**] [–**p**] [–**s**] [–**w**] [–**d** *dest*] [–**f** *form-name* [–**d** **any**]]
[–**H** *special-handling*] [–**n** *number*] [–**o** *option*] [–**P** *page-list*]
[–**q** *priority-level*] [–**S** *character-set* [–**d** **any**]] [–**S** *print-wheel* [–**d** **any**]]
[–**t** *title*] [–**T** *content-type* [–**r**]] [–**y** *mode-list*]

cancel [*request-ID...*] [*printer...*]
cancel –**u** *login-ID-list* [*printer...*]

AVAILABILITY

SUNWlpu

DESCRIPTION

The first form of the **lp** command arranges for the named *file(s)* and associated information (collectively called a *request*) to be printed. If no file names are specified on the command line, the standard input is assumed. The standard input may be specified along with a named *file*(s) on the command line by listing the file name(s) and specifying '–' (dash) for the standard input. The *files* will be printed in the order in which they appear on the shell command line.

The LP print service associates a unique *request-ID* (with the –**i** option) with each request and displays it on the standard output. This *request-ID* can be used later with the –**i** option when canceling or changing a request, or when determining its status. (See the section on **cancel** for details about canceling a request, and **lpstat**(1) for information about checking the status of a print request.)

The second form of **lp** is used to change the options for a request. The print request identified by the *request-ID* is changed according to the printing options specified with this shell command. The printing options available are the same as those with the first form of the **lp** shell command. If the request has finished printing, the change is rejected. If the request is already printing, it will be stopped and restarted from the beginning (unless the –**P** option has been given).

The **cancel** command allows users to cancel print requests previously sent with the **lp** command. The first form of **cancel** permits cancellation of requests based on their *request-ID*. The second form of cancel permits cancellation of requests based on the *login-ID* of their owner.

Sending a Print Request

The first form of the **lp** command is used to send a print request to a particular printer or group of printers.

OPTIONS

Options to **lp** always precede any file names, but may be specified in any order. The following options are available for **lp :**

–c Make a copy of the *file* before printing. Normally, *file* will not be copied, but will be linked whenever possible. If the **–c** option is not given, then the user should be careful not to remove any *file* before the request has been printed in its entirety. It should also be noted that if the **–c** option is not specified, any changes made to the named *file* after the request is made but before it is printed will be reflected in the printed output.

–d *dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. If *dest* is **any**, then the request will be printed on any printer which can handle it. Under certain conditions, (unavailability of printers, file space limitations, and so on) requests for specific destinations may not be accepted (see **lpstat**(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see **lpstat**(1)).

–f *form-name* [**–d any**]
Print the request on the form *form-name*. The LP print service ensures that the form is mounted on the printer. If *form-name* is requested with a printer destination that cannot support the form, the request is rejected. If *form-name* has not been defined for the system, or if the user is not allowed to use the form, the request is rejected (see **lpforms**(1M)). When the **–d any** option is given, the request is printed on any printer that has the requested form mounted and can handle all other needs of the print request.

–H *special-handling*
Print the request according to the value of *special-handling*. Acceptable values for *special-handling* are defined below:

hold Do not print the request until notified. If printing has already begun, stop it. Other print requests will go ahead of a held request until it is resumed.

resume Resume a held request. If the request had begun to print when held, it will be the next request printed, unless it is superseded by an **immediate** request.

immediate (Available only to LP administrators.) Print the request next. If more than one request is assigned the most recent request is printed next. If a request is currently printing on the desired printer, a hold request must be issued to allow the immediate request to print.

−m Send mail (see **mail**(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n *number* Print *number* copies (default is **1**) of the output.

−o *option* Specify printer-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once (**−o** $option_1$ **−o** $option_2$ ... **−o** $option_n$), or by specifying the **−o** keyletter followed by a list of options enclosed in double quotes (that is, **−o** "$option_1$ $option_2$ ... $option_n$"). The standard interface recognizes the following options:

nobanner Do not print a banner page with this request. (The administrator can disallow this option at any time.)

nofilebreak
Do not insert a form feed between the files given, if submitting a job to print more than one file.

length=*scaled-decimal-number*
Print this request with pages *scaled-decimal-number* lines long. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter ‘‘i’’ for inches, or the letter ‘‘c’’ for centimeters. For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with ‘‘i’’). For example, **length=66** indicates a page length of **66** lines, **length=11i** indicates a page length of **11** inches, and **length=27.94c** indicates a page length of **27.94** centimeters.

This option may not be used with the –**f** option.

width=*scaled-decimal-number*
Print this request with page-width set to *scaled-decimal-number* columns wide. (See the explanation of *scaled-decimal-numbers* in the discussion of **length**, above.) This option may not be used with the –**f** option.

lpi=*scaled-decimal-number*
Print this request with the line pitch set to *scaled-decimal-number* lines per inch. This option may not be used with the −**f** option.

cpi=*scaled-decimal-number*
Print this request with the character pitch set to *scaled-decimal-number* characters per inch. Character pitch can also be set to **pica** (representing **10** characters per inch) or **elite** (representing **12** characters per inch), or it can be **compressed** (representing as many characters as a printer can handle). There is no standard number of characters per inch for all printers; see the Terminfo database (see **terminfo**(4)) for the default character pitch for your printer.

This option may not be used with the −**f** option.

stty=’*stty-option-list*’
A list of options valid for the **stty** command; enclose the list with single quotes if it contains blanks.

−**P** *page-list* Print the pages specified in *page-list*. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

The *page-list* may consist of range(s) of numbers, single page numbers, or a combination of both. The pages will be printed in ascending order.

−**p** Enable notification on completion of the print request. Delivery of the notification is dependent on additional software.

−**q** *priority-level* Assign this request *priority-level* in the printing queue. The values of *priority-level* range from **0**, the highest priority, to **39**, the lowest priority. If a priority is not specified, the default for the print service is used, as assigned by the system administrator. A priority limit may be assigned to individual users by the system administrator.

−**s** Suppress messages from **lp** such as those that begin with "**request id is** . . ."

−**S** *character-set* [−**d any**]
−**S** *print-wheel* [−**d any**]
Print this request using the specified *character-set* or *print-wheel*. If a form was requested and it requires a character set or print wheel other than the one specified with the −**S** option, the request is rejected.

For printers that take print wheels: if the print wheel specified is not one listed by the administrator as acceptable for the printer specified in this request, the request is rejected unless the print wheel is already mounted on the printer.

For printers that use selectable or programmable character sets: if the *character-set* specified is not one defined in the Terminfo database for the printer (see **terminfo**(4)), or is not an alias defined by the administrator, the request is rejected.

When the **−d any** option is used, the request is printed on any printer that has the print wheel mounted or any printer that can select the character set, and that can handle the needs of the request.

−t *title* Print *title* on the banner page of the output. If *title* is not supplied the name of the file is printed on the banner page. Enclose *title* in quotes if it contains blanks.

−T *content-type* [**−r**]
Print the request on a printer that can support the specified *content-type*. If no printer accepts this type directly, a filter will be used to convert the content into an acceptable type. If the **−r** option is specified, a filter will not be used. If **−r** is specified, and no printer accepts the *content-type* directly, the request is rejected. If the *content-type* is not acceptable to any printer, either directly or with a filter, the request is rejected.

−w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

−y *mode-list* Print this request according to the printing modes listed in *mode-list*. The allowed values for *mode-list* are locally defined. This option may be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

Canceling a Print Request

The **cancel** command cancels requests for print jobs made with the **lp** command. The first form allows a user to specify one or more *request-ID* of print jobs to be canceled. Alternatively, the user can specify one or more *printer*, on which only the currently printing job will be canceled.

The second form of **cancel** permits a user to cancel all of his or her own jobs on all printers. In this form the *printer* option can be used to restrict the printer(s) on which the user's job(s) will be canceled. Note: In this form, when the *printer* option is used, all jobs queued for that printer will be canceled. A printer class is not a valid argument.

Users without special privileges can cancel only requests associated with their own login IDs. The system administrator can cancel jobs submitted by any user. The *login-ID-list* must be enclosed in quotes if it contains blanks.

For printers that take mountable print wheels or font cartridges, if you do not specify a particular print wheel or font with the **−S** option, the one mounted at the time your request is printed will be used. Use the **lpstat −p** *printer* **−l** command to see which print wheels are available on a particular printer, or the **lpstat −S −l** command to find out what print wheels are available and on which printers. For printers that have selectable character sets, you will get the standard character set if you don't use the **−S** option.

FILES

/var/spool/lp/∗

SEE ALSO

enable(1), **lpstat**(1), **mail**(1), **postprint**(1), **pr**(1), **accept**(1M), **lpadmin**(1M), **lpfilter**(1M), **lpforms**(1M), **lpsched**(1M), **lpsystem**(1M), **lpusers**(1M), **terminfo**(4)

NOTES

Printers for which requests are not being accepted will not be considered when the **lp** command is run and the destination is **any**. (Use the **lpstat** −**a** command to see which printers are accepting requests.) On the other hand, if a request is destined for a class of printers and the class itself is accepting requests, then *all* printers in the class will be considered, regardless of their acceptance status.

NAME

lpc – line printer control program

SYNOPSIS

/usr/ucb/lpc [*command* [*parameter*...]]

AVAILABILITY

SUNWscpu

DESCRIPTION

lpc controls the operation of the printer, or of multiple printers. **lpc** commands can be used to start or stop a printer, disable or enable a printer's spooling queue, rearrange the order of jobs in a queue, or display the status of each printer—along with its spooling queue and printer daemon.

With no arguments, **lpc** runs interactively, prompting with '**lpc>**'. If arguments are supplied, **lpc** interprets the first as a *command* to execute; each subsequent argument is taken as a *parameter* for that command. The standard input can be redirected so that **lpc** reads commands from a file.

USAGE

Commands

Commands may be abbreviated to an unambiguous substring. Note: the *printer* parameter is specified just by the name of the printer (as **lw**), not as you would specify it to **lpr**(1B) or **lpq**(1B) (not as –**Plw**).

? [*command*]...
help [*command*]...

Display a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort [**all** | [*printer* ...]]

Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by **lpr**(1B)) for the specified printers. The **abort** command can only be used by the privileged user.

clean [**all** | [*printer* ...]]

Remove all files created in the spool directory by the daemon from the specified printer queue(s) on the local machine. The **clean** command can only be used by the privileged user.

disable [**all** | [*printer* ...]]

Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by **lpr**(1B). The **disable** command can only be used by the privileged user.

down [**all** | [*printer* ...]] [*message*]

Turn the specified printer queue off, disable printing and put *message* in the printer status file. The message does not need to be quoted, the remaining arguments are treated like **echo**(1). This is normally used to take a printer down and let others know why (**lpq**(1B) indicates that the printer is down, as does the **status** command).

enable [**all** | [*printer* . . .]]
Enable spooling on the local queue for the listed printers, so that **lpr**(1B) can put new jobs in the spool queue. The **enable** command can only be used by the privileged user.

exit
quit Exit from **lpc**.

restart [**all** | [*printer* . . .]]
Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. This command can be run by any user.

start [**all** | [*printer* . . .]]
Enable printing and start a spooling daemon for the listed printers. The **start** command can only be used by the privileged user.

status [**all** | [*printer* . . .]]
Display the status of daemons and queues on the local machine. This command can be run by any user.

stop [**all** | [*printer* . . .]]
Stop a spooling daemon after the current job completes and disable printing. The **stop** command can only be used by the privileged user.

topq *printer* [*job#* . . .] [*user* . . .]
Move the print job(s) specified by *job#* or those job(s) belonging to *user* to the top (head) of the printer queue. The **topq** command can only be used by the privileged user.

up [**all** | [*printer* . . .]] Enable everything and start a new printer daemon. Undoes the effects of **down**.

FILES

/var/spool/lp/∗
/var/spool/lp/system/pstatus

SEE ALSO

echo(1), **lpq**(1B), **lpr**(1B), **lprm**(1B), **lpsched**(1M)

DIAGNOSTICS

?Ambiguous command
The abbreviation you typed matches more than one command.

?Invalid command
You typed a command or abbreviation that was not recognized.

?Privileged command
You used a command can be executed only by the privileged user.

lpc: *printer* **: unknown printer to the print service**
The **printer** was not found in the LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use '**lptstat −p**' to find the reason.

lpc: error on opening queue to spooler
The connection to **lpsched** on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon **/usr/lib/lp/lpsched** is running.

lpc: Can't send message to LP print service

lpc: Can't receive message from LP print service
These indicate that the LP print service has been stopped. Get help from the system administrator.

lpc: Received unexpected message from LP print service
It is likely there is an error in this software. Get help from system administrator.

NAME

lpq – display the queue of printer jobs

SYNOPSIS

/usr/ucb/lpq [**–P** *printer*] [**–l**] [**+** [*interval*]] [*job#* . . .] [*username* . . .]

AVAILABILITY

SUNWscpu

DESCRIPTION

lpq displays the contents of a printer queue. It reports the status of jobs specified by *job#*, or all jobs owned by the user specified by *username*. **lpq** reports on all jobs in the default printer queue when invoked with no arguments.

For each print job in the queue, **lpq** reports the user's name, current position, the names of input files comprising the job, the job number (by which it is referred to when using **lprm**(1B)) and the total size in bytes. Normally, only as much information as will fit on one line is displayed. Jobs are normally queued on a first-in-first-out basis. Filenames comprising a job may be unavailable, such as when **lpr** is used at the end of a pipeline; in such cases the filename field indicates the standard input.

If **lpq** warns that there is no daemon present (that is, due to some malfunction), the **lpc**(1B) command can be used to restart a printer daemon.

OPTIONS

–P *printer*	Display information about the queue for the specified *printer*. In the absence of the **–P** option, the queue to the printer specified by the **PRINTER** variable in the environment is used. If the **PRINTER** variable is not set, and the **LPDEST** environment variable is not set, the queue for the default printer is used.
–l	Display queue information in long format; includes the name of the host from which the job originated.
+[*interval*]	Display the spool queue periodically until it empties. This option clears the terminal screen before reporting on the queue. If an *interval* is supplied, **lpq** sleeps that number of seconds in between reports.

FILES

/var/spool/lp	spooling directory
/var/spool/lp/tmp/*system_name***/**∗**-0**	request files specifying jobs

SEE ALSO

lp(1), **lpr**(1B), **lprm**(1B), **lpc**(1B), **lpsched**(1M)

DIAGNOSTICS

printer **is printing**

The **lpq** program queries the spooler **LPSCHED** about the status of the printer. If the printer is disabled, the superuser can restart the spooler using **lpc**(1B).

printer **waiting for auto-retry (offline ?)**

The daemon could not open the printer device. The printer may be turned offline. This message can also occur if a printer is out of paper, the paper is jammed, and so on. Another possible cause is that a process, such as an output filter, has exclusive use of the device. The only recourse in this case is to kill the offending process and restart the printer with **lpc**.

waiting for *host* **to come up**
A daemon is trying to connect to the remote machine named *host*, in order to send the files in the local queue. If the remote machine is up, **lpd** on the remote machine is probably dead or hung and should be restarted using **lpc**.

sending to *host*
The files are being transferred to the remote *host*, or else the local daemon has hung while trying to transfer the files.

printer disabled reason:
The printer has been marked as being unavailable with **lpc**.

lpq: The LP print service isn't running or can't be reached.
The **lpsched** process overseeing the spooling queue does not exist. This normally occurs only when the daemon has unexpectedly died. You can restart the printer daemon with **lpc**.

lpr: *printer* **: unknown printer**
The **printer** was not found in the System V LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use '**lptstat –p**' to find the reason.

lpr: error on opening queue to spooler
The connection to **lpsched** on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon **/usr/lib/lpsched** is running.

lpr: Can't send message to LP print service

lpr: Can't receive message from LP print service
These indicate that the LP print service has been stopped. Get help from the system administrator.

lpr: Received unexpected message from LP print service
It is likely there is an error in this software. Get help from system administrator.

NOTES

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

NAME

lpr – send a job to the printer

SYNOPSIS

/usr/ucb/lpr [–**P** *printer*] [–**#** *copies*] [–**C** *class*] [–**J** *job*] [–**T** *title*]
[–**i** [*indent*]] [–**w** *cols*] [–**B**] [–**m**] [–**h**] [–**s**]
[–*filter_option*] [*filename* . . .]

AVAILABILITY

SUNWscpu

DESCRIPTION

lpr forwards printer jobs to a spooling area for subsequent printing as facilities become available. Each printer job consists of copies of, or, with –**s ,** complete pathnames of each *filename* you specify. The spool area is managed by the line printer spooler, **lpsched**. **lpr** reads from the standard input if no files are specified.

OPTIONS

–**P** *printer* Send output to the named *printer*. In the absence of the –**P** option, the queue to the printer specified by the **PRINTER** variable in the environment is used. If the **PRINTER** variable is not set, and the **LPDEST** environment variable is not set, the queue for the default printer is used.

–**#** *copies* Produce the number of *copies* indicated for each named file. For example:

```
lpr –#3 index.c lookup.c
```

produces three copies of **index.c**, followed by three copies of **lookup.c**. On the other hand,

```
cat index.c lookup.c | lpr –#3
```

generates three copies of the concatenation of the files.

–**C** *class* Print *class* as the job classification on the burst page. For example,

```
lpr –C Operations new.index.c
```

replaces the system name (the name returned by *hostname*) with **Operations** on the burst page, and prints the file **new.index.c**.

–**J** *job* Print *job* as the job name on the burst page. Normally, **lpr** uses the first file's name.

–**T** *title* Use *title* instead of the file name for the title used by **pr**(1).

–**i**[*indent*] Indent output *indent* SPACE characters. Eight SPACE characters is the default.

–**w** *cols* Use *cols* as the page width for **pr**.

–**m** Send mail upon completion.

–**h** Suppress printing the burst page.

–**s** Use the full pathnames (not symbolic links) of the files to be printed rather than trying to copy them. This means the data files should not be modified or removed until they have been printed. –**s** only prevents copies of local files from being made. Jobs from remote hosts are copied anyway. **–s** only works with named data files; if the **lpr** command is at the end of a pipeline, the data is copied to the spool.

filter_option The following single letter options notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

–p	Use **pr** to format the files (**lpr** **–p** is very much like **pr | lpr**).
–l	Print control characters and suppress page breaks.
–t	The files contain **troff**(1) (cat phototypesetter) binary data.
–n	The files contain data from **ditroff** (device independent troff).
–d	The files contain data from **tex** (DVI format from Stanford).
–g	The files contain standard plot data as produced by the **plot**(1B) routines.
–v	The files contain a raster image. The printer must support an appropriate imaging model such as PostScript® in order to print the image.
–c	The files contain data produced by *cifplot*.
–f	Interpret the first character of each line as a standard FORTRAN carriage control character.

If no *filter_option* is given (and the printer can interpret PostScript), the string '**%!**' as the first two characters of a file indicates that it contains PostScript commands.

These filter options offer a standard user interface, and all options may not be available for, nor applicable to, all printers.

FILES

/etc/passwd	personal identification
/usr/lib/lp/lpsched	System V line printer spooler
/var/spool/lp/tmp/∗	directories used for spooling
/var/spool/lp/tmp/*system*/∗**-0**	spooler control files
/var/spool/lp/tmp/*system*/∗-*N*	(*N* is an integer and > 0) data files specified in '∗**-0**' files

SEE ALSO

lp(1), **lpc**(1B) **lpq**(1B), **lprm**(1B), **plot**(1B), **pr**(1), **troff**(1), **lpsched**(1M)

DIAGNOSTICS

lpr: *printer*: **unknown printer**

The **printer** was not found in the LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use '**lptstat** **–p**' to find the reason.

lpr: **error on opening queue to spooler**

The connection to **lpsched** on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon **/usr/lib/lpsched** is running.

lpr: *printer*: **printer queue is disabled**

This means the queue was turned off with

```
/usr/etc/lpc disable printer
```

to prevent **lpr** from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a privileged user with **lpc**.

lpr: Can't send message to the LP print service

lpr: Can't receive message from the LP print service

These indicate that the LP print service has been stopped. Get help from the system administrator.

lpr: Received unexpected message from LP print service

It is likely there is an error in this software. Get help from system administrator.

lpr: There is no filter to convert the file content

Use the '**lpstat –p –l**' command to find a printer that can handle the file type directly, or consult with your system administrator.

lpr: cannot access the file

Make sure file names are valid.

NOTES

lp is the preferred interface.

Command-line options cannot be combined into a single argument as with some other commands. The command:

```
lpr –fs
```

is not equivalent to

```
lpr –f –s
```

Placing the **–s** flag first, or writing each option as a separate argument, makes a link as expected.

lpr –p is not precisely equivalent to **pr | lpr**. **lpr –p** puts the current date at the top of each page, rather than the date last modified.

Fonts for **troff**(1) and T_EX® reside on the printer host. It is currently not possible to use local font libraries.

lpr objects to printing binary files.

The **–s** option, intended to use symbolic links in SunOS, does not use symbolic links in the compatibility package. Instead, the complete path names are used. Also, the copying is avoided only for print jobs that are run from the printer host itself. Jobs added to the queue from a remote host are always copied into the spool area. That is, if the printer does not reside on the host that **lpr** is run from, the spooling system makes a copy the file to print, and places it in the spool area of the printer host, regardless of **–s**.

NAME

lprm – remove jobs from the printer queue

SYNOPSIS

/usr/ucb/lprm [–**P***printer*] [–] [*job #* . . .] [*username* . . .]

AVAILABILITY

SUNWscpu

DESCRIPTION

lprm removes a job or jobs from a printer's spooling queue. Since the spool directory is protected from users, using **lprm** is normally the only method by which a user can remove a job.

Without any arguments, **lprm** deletes the job that is currently active, provided that the user who invoked **lprm** owns that job.

When the privileged user specifies a *username*, **lprm** removes all jobs belonging to that user.

You can remove a specific job by supplying its job number as an argument, which you can obtain using **lpq**(1B). For example:

```
example% lpq –Phost
host is ready and printing
Rank Owner   Job     Files   Total Size
active       wendy   385     standard input  35501 bytes
example% lprm –Phost 385
```

lprm reports the names of any files it removes, and is silent if there are no applicable jobs to remove.

lprm Sends the request to cancel a job to the print spooler, **LPSCHED**.

OPTIONS

–**P***printer* Specify the queue associated with a specific printer. Otherwise the value of the **PRINTER** variable in the environment is used. If the **PRINTER** variable is not set, and the **LPDEST** environment variable is not set, the queue for the default printer is used.

– Remove all jobs owned by you. If invoked by the privileged user, all jobs in the spool are removed. Job ownership is determined by the user's login name and host name on the machine where the **lpr** command was executed.

FILES

/var/spool/lp/∗	spooling directories

SEE ALSO

lp(1), **lpq**(1B), **lpr**(1B), **lpsched**(1M)

DIAGNOSTICS

lprm: *printer* **: unknown printer**

The **printer** was not found in the System V LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use '**lptstat –p**' to find the reason.

lprm: error on opening queue to spooler
The connection to **lpsched** on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon **/usr/lib/lpsched** is running.

lprm: Can't send message to the LP print service

lprm: Can't receive message from the LP print service
These indicate that the LP print service has been stopped. Get help from the system administrator.

lprm: Received unexpected message from the LP print service
It is likely there is an error in this software. Get help from system administrator.

lprm: Can't cancel request
You are not allowed to remove another's request.

NOTES

An active job may be incorrectly identified for removal by an **lprm** command issued with no arguments. During the interval between an **lpq**(1B) command and the execution of **lprm**, the next job in queue may have become active; that job may be removed unintentionally if it is owned by you. To avoid this, supply **lprm** with the job number to remove when a critical job that you own is next in line.

Only the privileged user can remove print jobs submitted from another host.

lp is the preferred interface.

NAME

lpstat – print information about the status of the LP print service

SYNOPSIS

lpstat [–**d**] [–**r**] [–**R**] [–**s**] [–**t**] [–**a** [*list*]] [–**c** [*list*]] [–**f** [*list*] [–**l**]]
[–**o** [*list*]] [–**p** [*list*] [–**D**] [–**l**]] [–**P**] [–**S** [*list*] [–**l**]] [–**u** [*login-ID-list*]]
[–**v** [*list*]]

AVAILABILITY

SUNWlpu

DESCRIPTION

The **lpstat** command prints information about the current status of the LP print service.

If no options are given, then **lpstat** prints the status of all the user's print requests made by **lp** (see **lp**(1)). Any arguments that are not *options* are assumed to be *request-IDs* as returned by **lp**. The **lpstat** command prints the status of such requests. The *options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items separated from one another by spaces enclosed in quotes. For example:

```
example% lpstat –u "user1 user2 user3"
```

Specifying **all** after any keyletter that takes *list* as an argument causes all information relevant to the keyletter to be printed. For example, the command:

```
example% lpstat –o all
```

prints the status of all output requests.

The omission of a *list* following such key letters causes all information relevant to the key letter to be printed. For example, the command:

```
example% lpstat –o
```

prints the status of all output requests.

OPTIONS

–**a** [*list*]	Reports whether print destinations are accepting requests. *list* is a list of intermixed printer names and class names.
–**c** [*list*]	Print name of all classes and their members. *list* is a list of class names.
–**d**	Print the system default destination for output requests.
–**f** [*list*] [–**l**]	Print a verification that the forms in *list* are recognized by the LP print service. *list* is a list of forms; the default is **all**. The –**l** option will list the form descriptions.
–**o** [*list*]	Print the status of output requests: *list* is a list of intermixed printer names, class names, and *request-IDs*. The keyletter –**o** may be omitted.
–**p** [*list*] [–**D**] [–**l**]	Print the status of printers. *list* is a list of printer names. If the –**D** option is given, a brief description is printed for each printer in *list*. If the –**l** option is given, and the printer is on the local machine, a full description of each printer's configuration is given, including the form mounted, the acceptable

content and printer types, a printer description, the interface used, and so on.

–P Print the paper types.

–r Print the status of the LP request scheduler.

–R Print a number showing the position of the job in the print queue.

–s Print a status summary, including the status of the LP scheduler, the system default destination, a list of class names and their members, a list of printers and their associated devices, a list of the machines sharing print services, a list of all forms currently mounted, and a list of all recognized character sets and print wheels.

–S [*list*] [–**l**]
Print a verification that the character sets or the print wheels specified in *list* are recognized by the LP print service. Items in *list* can be character sets or print wheels; the default for the list is **all**. If the –**l** option is given, each line is appended by a list of printers that can handle the print wheel or character set. The list also shows whether the print wheel or character set is mounted, or specifies the built-in character set into which it maps.

–t Print all status information. This includes all the information obtained with the –**s** option, plus the acceptance and idle / busy status of all printers.

–u [*login-ID-list*]
Print the status of output requests for users. The *login-ID-list* argument may include any or all of the following constructs:

login-ID	a user on any system
system_name!*login-ID*	a user on system *system_name*
system_name!**all**	all users on system *system_name*
all!*login-ID*	a user on all systems
all	all users on all systems

–v [*list*] Print the names of printers and the path names of the devices associated with them or remote system names for network printers: *list* is a list of printer names.

FILES

/etc/lp/∗
/var/spool/lp/∗

ENVIRONMENT

Date and time format is based on locale specified by the **LC_ALL**, **LC_TIME** or **LANG** environments, in that order of priority.

SEE ALSO

enable(1), **lp**(1)

NAME

lptest – generate lineprinter ripple pattern

SYNOPSIS

/usr/ucb/lptest [*length* [*count*]]

AVAILABILITY

SUNWscpu

DESCRIPTION

lptest writes the traditional ‘‘ripple test’’ pattern on standard output. In 96 lines, this pattern will print all 96 printable ASCII characters in each position. While originally created to test printers, it is quite useful for testing terminals, driving terminal ports for debugging purposes, or any other task where a quick supply of random data is needed.

The *length* argument specifies the output line length if the the default length of 79 is inappropriate.

The *count* argument specifies the number of output lines to be generated if the default count of 200 is inappropriate.

NOTES

if *count* is to be specified, *length* must be also be specified.

This command is obsolete.

NAME

ls – list contents of directory

SYNOPSIS

ls [–**abcCdfFgilLmnopqrRstux1**] [*names*]

AVAILABILITY

SUNWcsu

DESCRIPTION

For each directory argument, **ls** lists the contents of the directory; for each file argument, **ls** repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format for output directed to a terminal is multi–column with entries sorted down the columns. The –**1** option allows single column output and –**m** enables stream output format. In order to determine output formats for the –**C**, –**x**, and –**m** options, **ls** uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the **terminfo**(4) database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

The mode printed under the –**l** option consists of ten characters. The first character may be one of the following:

d	the entry is a directory;
l	the entry is a symbolic link;
b	the entry is a block special file;
c	the entry is a character special file;
p	the entry is a fifo (a.k.a. ‘‘named pipe’’) special file;
–	the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner’s permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, ‘‘execute’’ permission is interpreted to mean permission to search the directory for a specified file.

ls –l (the **l**ong list) prints its output as follows:

```
–rwxrwxrwx   1  smith    dev       10876    May  16  9:42  part2
```

Reading from right to left, you see that the current directory holds one file, named **part2**. Next, the last time that file’s contents were modified was 9:42 A.M. on May 16. The file contains 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group **dev** (perhaps indicating ‘‘development’’), and his or her login name is **smith**. The number, in this case **1**, indicates the number of links to file **part2**; see **cp**(1). Finally, the dash and letters tell you that user, group, and others have permissions to read, write, and execute **part2**.

The execute (**x**) symbol here occupies the third position of the three-character sequence. A – in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

r	the file is readable
w	the file is writable
x	the file is executable
–	the indicated permission is *not* granted
l	mandatory locking occurs during access (the set-group- ID bit is on and the group execution bit is off)
s	the **s**et-user-ID or **s**et-group-ID bit is on, and the corresponding user or group execution bit is also on
S	undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
t	the 1000 (octal) bit, or sticky bit, is on (see **chmod**(1)), and execution is on
T	the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than **x** or –. **s** also may occupy this position, referring to the state of the **s**et-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user you login as.

In the case of the sequence of group permissions, **l** may occupy the third position. **l** refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

OPTIONS

–a	List all entries, including those that begin with a dot (**.**), which are normally not listed.
–b	Force printing of non-printable characters to be in the octal \ddd notation.
–c	Use time of last modification of the i-node (file created, mode changed, and so forth) for sorting (**–t**) or printing (**–l**).
–C	Multi-column output with entries sorted down the columns. This is the default output format.
–d	If an argument is a directory, list only its name (not its contents); often used with **–l** to get the status of a directory.
–f	Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **–l**, **–t**, **–s**, and **–r**, and turns on **–a**; the order is the order in which entries appear in the directory.
–F	Put a slash (**/**) after each filename if the file is a directory, an asterisk (*) if the file is an executable, and an at-sign (**@**) if the file is a symbolic link.
–g	The same as **–l**, except that the owner is not printed.
–i	For each file, print the i-node number in the first column of the report.

–l	List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see above). If the file is a special file, the size field instead contains the major and minor device numbers rather than a size. If the file is a symbolic link, the filename is printed followed by "→" and the pathname of the referenced file.
–L	If an argument is a symbolic link, list the file or directory the link references rather than the link itself.
–m	Stream output format; files are listed across the page, separated by commas.
–n	The same as **–l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.
–o	The same as **–l**, except that the group is not printed.
–p	Put a slash (**/**) after each filename if the file is a directory.
–q	Force printing of non-printable characters in file names as the character question mark (**?**).
–r	Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
–R	Recursively list subdirectories encountered.
–s	Give size in blocks, including indirect blocks, for each entry.
–t	Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **–n** and **–c**.)
–u	Use time of last access instead of last modification for sorting (with the **–t** option) or printing (with the **–l** option).
–x	Multi-column output with entries sorted across rather than down the page.
–1	Print one entry per line of output.

EXAMPLES

An example of a file's permissions is:

–rwxr––r––

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions is:

–rwsr–xr–x

This describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

Another example of a file's permissions is:

–rw–rwl–––

This describes a file that is readable and writable only by the user and the group and can be locked during access.

An example of a command line:

```
example% ls −a
```

This command prints the names of all files in the current directory, including those that begin with a dot (**.**), which normally do not print.

Another example of a command line:

```
example% ls −aisn
```

This command provides information on **a**ll files, including those that begin with a dot (**a**), the **i**-number—the memory address of the i-node associated with the file—printed in the left-hand column (**i**); the **s**ize (in blocks) of the files, printed in the column to the right of the i-numbers (**s**); finally, the report is displayed in the **n**umeric version of the long list, printing the **UID** (instead of user name) and **GID** (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **ls** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **ls** behaves.

LC_CTYPE
Determines how **ls** handles characters. When **LC_CTYPE** is set to a valid value, **ls** can display and handle text and filenames containing valid characters for that locale. **ls** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **ls** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME
Determines how **ls** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/etc/group	group IDs for **ls −l** and **ls −g**
/etc/passwd	user IDs for **ls −l** and **ls −o**
/usr/share/lib/terminfo/?/∗	terminal information database

SEE ALSO

chmod(1), **cp**(1), **find**(1), **terminfo**(4), **environ**(5)

NOTES

Unprintable characters in file names may confuse the columnar output options.

The total block count will be incorrect if if there are hard links among the files.

NAME

ls – list the contents of a directory

SYNOPSIS

/usr/ucb/ls [**–aAcCdfFgilLqrRstu1**] *filename* . . .

AVAILABILITY

SUNWscpu

DESCRIPTION

For each *filename* which is a directory, **ls** lists the contents of the directory; for each *filename* which is a file, **ls** repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

Permissions Field

The mode printed under the **–l** option contains 10 characters interpreted as follows. If the first character is:

d entry is a directory;
b entry is a block-type special file;
c entry is a character-type special file;
l entry is a symbolic link;
p entry is a FIFO (also known as “named pipe”) special file;
s entry is an **AF_UNIX** address family socket, or
– entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next refers to permissions to others in the same user-group; and the last refers to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, “execute” permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

r the file is readable;
w the file is writable;
x the file is executable;
– the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the owner-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally **x** or ‘–’) is **true** if the 1000 bit of the mode is on. See **chmod**(1) for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

OPTIONS

–a	List all entries; in the absence of this option, entries whose names begin with a ‘**.**’ are *not* listed (except for the privileged user, for whom **ls** normally prints even files that begin with a ‘**.**’).
–A	Same as –**a**, except that ‘**.**’ and ‘**. .**’ are not listed.
–c	Use time of last edit (or last mode change) for sorting or printing.
–C	Force multi-column output, with entries sorted down the columns; for **ls**, this is the default when output is to a terminal.
–d	If argument is a directory, list only its name (not its contents); often used with –**l** to get the status of a directory.
–f	Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off –**l**, –**t**, –**s**, and –**r**, and turns on –**a**; the order is the order in which entries appear in the directory.
–F	Mark directories with a trailing slash (‘**/**’), executable files with a trailing asterisk (‘*’), symbolic links with a trailing at-sign (‘**@**’), and **AF_UNIX** address family sockets with a trailing equals sign (‘**=**’).
–g	For **ls**, show the group ownership of the file in a long output.
–i	For each file, print the i-node number in the first column of the report.
–l	List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. If the file is a special file the size field will instead contain the major and minor device numbers. If the time of last modification is greater than six months ago, it is shown in the format ‘*month date year*’; files modified within six months show ‘*month date time*’. If the file is a symbolic link the pathname of the linked-to file is printed preceded by ‘**—>**’.
–L	If argument is a symbolic link, list the file or directory the link references rather than the link itself.
–q	Display non-graphic characters in filenames as the character **?**; for **ls**, this is the default when output is to a terminal.
–r	Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
–R	Recursively list subdirectories encountered.
–s	Give size of each file, including any indirect blocks used to map the file, in kilobytes.
–t	Sort by time modified (latest first) instead of by name.
–u	Use time of last access instead of last modification for sorting (with the –**t** option) and / or printing (with the –**l** option).
–1	Force one entry per line output format; this is the default when output is not to a terminal.

FILES

/etc/group	to get group ID for ‘**ls –g**’
/etc/passwd	to get user ID’s for ‘**ls –l**’ and ‘**ls –o**’

NOTES

NEWLINE and TAB are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as ‘**ls –s**’ is much different than ‘**ls –s | lpr**’. On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

Unprintable characters in file names may confuse the columnar output options.

NAME

m4 – macro processor

SYNOPSIS

m4 [–**e**] [–**s**] [–**B***int*] [–**H***int*] [–**S***int*] [–**T***int*] [–**D***name* [=*val*]] [–**U***name*]
[*filename* . . .]

DESCRIPTION

The **m4** command is a macro processor intended as a front end for C, assembler, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is – **,** the standard input is read. The processed text is written on the standard output.

Macro calls have the form:

name(*arg1*,*arg2*, . . ., *argn*)

The **(** must immediately follow the name of the macro. If the name of a defined macro is not followed by a **(,** it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphanumeric characters and underscore (**_**), where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses that happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

OPTIONS

The options and their effects are as follows:

–**e**	Operate interactively. Interrupts are ignored and the output is unbuffered.
–**s**	Enable line sync output for the C preprocessor (**#**line . . .)
–**B***int*	Change the size of the push-back and argument collection buffers from the default of 4,096.
–**H***int*	Change the size of the symbol table hash array from the default of 199. The size should be prime.
–**S***int*	Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
–**T***int*	Change the size of the token buffer from the default of 512 bytes.

To be effective, the above flags must appear before any file names and before any –**D** or –**U** flags:

–**D***name*[**=***val*]	Defines *name* to *val* or to null in *val*'s absence.
–**U***name*	Undefines *name*.

USAGE

m4 makes available the following built-in macros. These macros may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define	The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **$**_n_ in the replacement text, where _n_ is a digit, is replaced by the _n_-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is like **$***, but each argument is quoted (with the current quotes).
undefine	Removes the definition of the macro named in its argument.
defn	Returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
pushdef	Like **define**, but saves any previous definition.
popdef	Removes current definition of its argument(s), exposing the previous one, if any.
ifdef	If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word **unix** is predefined.
shift	Returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
changequote	Change quote symbols to the first and second arguments. The symbols may be up to five characters long. **changequote** without arguments restores the original values (that is, ` ´).
changecom	Change left and right comment markers from the default **#** and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
divert	**m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The **divert** macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
undivert	This macro causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
divnum	Returns the value of the current output stream.
dnl	Reads and discards characters up to and including the next new-line.

ifelse	This macro has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	Returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
decr	Returns the value of its argument decremented by 1.
eval	Evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include **+**, –, *, **/**, **%**, ** (exponentiation), bitwise **&**, | , ˆ, and ˜; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	Returns the number of characters in its argument.
index	Returns the position in its first argument where the second argument begins (zero origin), or –1 if the second argument does not occur.
substr	Returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	Transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	Returns the contents of the file named in the argument.
sinclude	This macro is identical to **include**, except that it says nothing if the file is inaccessible.
syscmd	This macro executes the command given in the first argument. No value is returned.
sysval	This macro is the return code from the last call to **syscmd**.
maketemp	Fills in a string of **XXXXX** in its argument with the current process ID.
m4exit	This macro causes immediate exit from **m4**. Argument 1, if given, is the exit code; the default is 0.
m4wrap	Argument 1 will be pushed back at final EOF; example: **m4wrap(`cleanup( )´)**
errprint	Prints its argument on the diagnostic output file.
dumpdef	Prints current names and definitions, for the named items, or for all if no arguments are given.

traceon	This macro with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	Turns off trace globally and for any macros specified. Macros specifically traced by **traceon** can be untraced only by specific calls to **traceoff**.

SEE ALSO

as(1)

NAME

mach – display the processor type of the current host

SYNOPSIS

/usr/ucb/mach

AVAILABILITY

SUNWscpu

DESCRIPTION

The **mach** command displays the processor-type of the current host.

SEE ALSO

arch(1B), **machid**(1), **uname**(1), **sysinfo**(2), **uname**(2)

NAME

machid, sun, iAPX286, i286, i386, i486, i860, pdp11, sparc, u3b, u3b2, u3b5, u3b15, vax, u370 – get processor type truth value

SYNOPSIS

sun
iAPX286
i386
pdp11
sparc
u3b
u3b2
u3b5
u3b15
vax
u370

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are using an instruction set that the command name indicates.

sun	True if you are on a Sun system.
iAPX286	True if you are on a computer using an iAPX286 processor.
i386	True if you are on a computer using an iAPX386 processor.
pdp11	True if you are on a PDP-11 / 45™ or PDP-11 / 70™.
sparc	True if you are on a computer using a SPARC-family processor.
u3b	True if you are on a 3B20 computer.
u3b2	True if you are on a 3B2 computer.
u3b5	True if you are on a 3B5 computer.
u3b15	True if you are on a 3B15 computer.
vax	True if you are on a VAX-11 / 750™ or VAX-11 / 780™.
u370	True if you are on an IBM® System / 370™ computer.

The commands that do not apply will return a false (non-zero) value. These commands are often used within makefiles (see **make**(1S)) and shell procedures (see **sh**(1)) to increase portability.

SEE ALSO

make(1S), **sh**(1), **test**(1), **true**(1), **uname**(1)

NOTES

The **machid** family of commands is obsolete. Use **uname –p** and **uname –m** instead.

NAME

mail, rmail – read mail or send mail to users

SYNOPSIS

Sending mail

mail [**–tw**] [**–m** *message_type*] *recipient*. . .

rmail [**–tw**] [**–m** *message_type*] *recipient*. . .

Reading mail

mail [**–ehpPqr**] [**–f** *filename*]

Debugging

mail [**–x** *debug_level*] [*other_mail_options*] *recipient*. . .

mail [**–T** *mailsurr_file*] *recipient*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

A *recipient* is usually a user name recognized by **login**(1). When *recipients* are named, **mail** assumes a message is being sent. It reads from the standard input up to an end-of-file (CTRL-D) or, if reading from a terminal device, until it reads a line consisting of just a period. When either of those indicators is received, **mail** adds the *letter* to the *mailfile* for each *recipient*.

A *letter* is composed of some *header lines* followed by a blank line followed by the *message content*. The *header lines* section of the letter consists of one or more UNIX postmarks:

From *sender date_and_time* [**remote from** *remote_system_name*]

followed by one or more standardized message header lines of the form:

keyword-name: [*printable text*]

where *keyword-name* is comprised of any printable, non-whitespace characters other than colon (‘:’). A **Content-Length:** header line, indicating the number of bytes in the *message content* will always be present unless the letter consists of only header lines with no message content. A **Content-Type:** header line that describes the type of the *message content* (such as text, binary, multipart, etc.) will also be present unless the letter consists of only header lines with no message content. Header lines may be continued on the following line if that line starts with white space.

OPTIONS

Sending mail

The following command-line arguments affect sending mail:

–m *message_type*
A **Message-Type:** line is added to the message header with the value of *message_type*.

–t A **To:** line is added to the message header for each of the intended recipients.

−w	A letter is sent to a remote recipient without waiting for the completion of the remote transfer program.

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If **mail** is interrupted during input, the message is saved in the file **dead.letter** to allow editing and resending. **dead.letter** is always appended to, thus preserving any previous contents. The initial attempt to append to (or create) **dead.letter** will be in the current directory. If this fails, **dead.letter** will be appended to (or created in) the user's login directory. If the second attempt also fails, no **dead.letter** processing will be done.

rmail only permits the sending of mail; **uucp**(1C) uses **rmail** as a security precaution. Any application programs that generate mail messages should be sure to invoke **rmail** rather than **mail** for message transport and/or delivery.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. There are numerous ways to address mail to recipients on remote systems depending on the transport mechanisms available to the local system. The two most prevalent addressing schemes are UUCP-style and Domain-style. With UUCP-style addressing, remote recipients are specified by prefixing the recipient name with the remote system name and an exclamation point, such as **sysa!user.** If **csh** is the default shell, **sysa\!user** should be used. A series of system names separated by exclamation points can be used to direct a letter through an extended network (such as **sysa!sysb!sysc!user** or **sysa\!sysb\!sysc\!user**). With Domain-style addressing, remote recipients are specified by appending an '**@**' and domain (and possibly sub-domain) information to the recipient name (such as **user@sf.att.com**). (The local System Administrator should be consulted for details on which addressing conventions are available on the local system.)

Reading Mail

The following command-line arguments affect reading mail:

−e	Mail is not printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
−h	A window of headers are initially displayed rather than the latest message. The display is followed by the **?** prompt.
−p	All messages are printed without prompting for disposition.
−P	All messages are printed with *all* header lines displayed, rather than the default selective header line display.
−q	**mail** terminates after interrupts. Normally an interrupt causes only the termination of the message being printed.
−r	Messages are printed in first-in, first-out order.
−f *filename*	**mail** uses *filename* (such as **mbox**) instead of the default *mailfile*.

mail, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. The default mode for printing messages is to display only those header lines of immediate interest. These include, but are not limited to, the UNIX **From** and **>From** postmarks, **From:**, **Date:**, **Subject:**, and **Content-Length:** header lines, and any recipient header lines such as **To:**, **Cc:**, **Bcc:**, and so forth. After the header lines have been displayed, **mail** will display the contents (body) of the message only if it contains no unprintable characters. Otherwise, **mail** will issue a warning statement about the message having binary content and **not** display the content. (This may be overridden via the **p** command. See below.)

For each message, the user is prompted with a **?** and a line is read from the standard input. The following commands are available to determine the disposition of the message:

#	Print the number of the current message.
–	Print previous message.
<new-line>,**+**, or **n**	Print the next message.
!*command*	Escape to the shell to do *command*.
a	Print message that arrived during the **mail** session.
d, or **dp**	Delete the current message and print the next message.
d *n*	Delete message number *n*. Do not go on to next message.
dq	Delete message and quit **mail**.
h	Display a window of headers around current message.
h *n*	Display a window of headers around message number *n*.
h a	Display headers of all messages in the user's *mailfile*.
h d	Display headers of messages scheduled for deletion.
m [*persons*]	Mail (and delete) the current message to the named *person*(*s*).
n	Print message number *n*.
p	Print current message again, overriding any indications of binary (that is, unprintable) content.
P	Override default brief mode and print current message again, displaying all header lines.
q, or CTRL-D	Put undeleted mail back in the *mailfile* and quit **mail**.
r [*users*]	Reply to the sender, and other *user(s)*, then delete the message.
s [*files*]	Save message in the named *file*(*s*) (**mbox** is default) and delete the message.
u [*n*]	Undelete message number *n* (default is last read).
w [*files*]	Save message contents, without any header lines, in the named *files* (**mbox** is default) and delete the message.
x	Put all mail back in the *mailfile* unchanged and exit **mail**.
y [*files*]	Same as save.
?	Print a command summary.

When a user logs in, the presence of mail, if any, is usually indicated. Also, notification is made if new mail arrives while using **mail**.

The permissions of *mailfile* may be manipulated using **chmod** in two ways to alter the function of **mail**. The other permissions of the file may be read-write (0666), read-only (0664), or neither read nor write (0660) to allow different levels of privacy. If changed to other than the default (mode 0660), the file will be preserved even when empty to perpetuate the desired permissions. (The administrator may override this file preservation using the **DEL_EMPTY_MAILFILE** option of **mailcnfg**.)

The group id of the mailfile must be **mail** to allow new messages to be delivered, and the mailfile must be writable by group **mail**.

Debugging

The following command-line arguments cause **mail** to provide debugging information:

–**T** *mailsurr_file*	**mail** displays how it will parse and interpret the **mailsurr** file.
–**x** *debug_level*	**mail** creates a trace file containing debugging information.

The –**T** option requires an argument that will be taken as the pathname of a test **mailsurr** file. If NULL (as in –**T** ""), the system **mailsurr** file will be used. To use, type '**mail** –**T** *test_file recipient*' and some trivial message (like "testing"), followed by a line with either just a dot ('.') or a CTRL-D. The result of using the –**T** option will be displayed on standard output and show the inputs and resulting transformations as **mailsurr** is processed by the **mail** command for the indicated recipient. Mail messages will never actually be sent or delivered when the –**T** option is used.

The –**x** option causes **mail** to create a file named **/tmp/MLDBG***process_id* that contains debugging information relating to how **mail** processed the current message. The absolute value of *debug_level* controls the verboseness of the debug information. Zero implies no debugging. If *debug_level* is greater than zero, the debug file will be retained **only** if **mail** encountered some problem while processing the message. If *debug_level* is less than zero the debug file will always be retained. The *debug_level* specified via –**x** overrides any specification of **DEBUG** in **/etc/mail/mailcnfg**. The information provided by the –**x** option is esoteric and is probably only useful to System Administrators. The output produced by the –**x** option is a superset of that provided by the –**T** option.

Delivery Notification

Several forms of notification are available for mail by including one of the following lines in the message header.

Transport-Options: [/*options*]

Default-Options: [/*options*]

>To: *recipient* [/*options*]

Where the ''/*options*'' may be one or more of the following:

/delivery	Inform the sender that the message was successfully delivered to the *recipient*'s mailbox.
/nodelivery	Do not inform the sender of successful deliveries.
/ignore	Do not inform the sender of **un**successful deliveries.
/return	Inform the sender if mail delivery fails. Return the failed message to the sender.
/report	Same as **/return** except that the original message is not returned.

The default is **/nodelivery/return**. If contradictory options are used, the first will be recognized and later, conflicting, terms will be ignored.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **mail** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **mail** behaves.

LC_CTYPE

Determines how **mail** handles characters. When **LC_CTYPE** is set to a valid value, **mail** can display and handle text and filenames containing valid characters for that locale. **mail** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **mail** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

FILES

dead.letter	unmailable text
/etc/passwd	to identify sender and locate recipients
/etc/mail/mailsurr	routing / name translation information
/etc/mail/mailcnfg	initialization information
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma∗	temporary file
/tmp/MLDBG∗	debug trace file
/var/mail/∗**.lock**	lock for mail directory
/var/mail/:saved	directory for holding temp files to prevent loss of data in the event of a system crash
/var/mail/*user*	incoming mail for *user*; that is, the *mailfile*

SEE ALSO

chmod(1), **login**(1), **mailx**(1), **vacation**(1), **write**(1), **uucp**(1C), **environ**(5)

Solaris Advanced User's Guide

NOTES

The interpretation and resulting action taken because of the header lines described in the Delivery Notifications section above will only occur if this version of **mail** is installed on the system where the delivery (or failure) happens. Earlier versions of **mail** may not support any types of delivery notification.

Conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

mailstats – print statistics collected by sendmail

SYNOPSIS

mailstats [*filename*]

AVAILABILITY

SUNWcsu

DESCRIPTION

mailstats prints out the statistics collected by the **sendmail**(1M) program on mailer usage. These statistics are collected if the file indicated by the **S** configuration option of **sendmail**(1M) (defined in **/etc/mail/sendmail.cf**), exists. The default statistics file is **/etc/mail/sendmail.st**. **mailstats** first prints the time that the statistics file was created and the last time it was modified. It will then print a table with one row for each mailer specified in the configuration file. The first column is the mailer number, followed by the symbolic name of the mailer. The next two columns refer to the number of messages received by **sendmail**, and the last two columns refer to messages sent by **sendmail**. The number of messages and their total size (in 1024 byte units) is given. No numbers are printed if no messages were sent (or received) for any mailer.

You might want to add an entry to **/var/spool/cron/crontabs/root** to reinitialize the statistics file once a night. Copy **/dev/null** into the statistics file or otherwise truncate it to reset the counters.

FILES

/dev/null	
/var/spool/cron/crontabs/root	
/etc/mail/sendmail.st	default statistics file.
/etc/mail/sendmail.cf	**sendmail**(1M) configuration file.

SEE ALSO

sendmail(1M)

NOTES

mailstats should read the configuration file instead of having a hard-wired table mapping mailer numbers to names.

NAME

mailx, mail, Mail – interactive message processing system

SYNOPSIS

mailx [–**BdeHiInNUvV~**] [–**f** [*filename* | +*folder*]] [–**T** *filename*] [–**u** *user*]
mailx [–**BdFintUv~**] [–**b** *bcc*] [–**c** *cc*] [–**h** *number*] [–**r** *address*] [–**s** *subject*]
recipient . . .

/usr/ucb/mail . . .

/usr/ucb/Mail . . .

AVAILABILITY

SUNWcsu

DESCRIPTION

The mail utilities listed above provide a comfortable, flexible environment for sending and receiving mail messages electronically. The **OPTIONS** and **USAGE** documented below for **mailx** also apply to **/usr/ucb/mail** and **/usr/ucb/Mail**, except where noted.

When reading mail, **mailx** provides commands to facilitate saving, deleting, and responding to messages. When sending mail, **mailx** allows editing, reviewing and other modification of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the **mailbox** for that user. When **mailx** is called to read messages, the **mailbox** is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the **mbox** and is normally located in the user's HOME directory (see **MBOX** in **Environment Variables** for a description of this file). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the –**f** option of the **mailx** command. Messages in the secondary file can then be read or otherwise processed using the same **Commands** as in the primary **mailbox**. This gives rise within these pages to the notion of a current **mailbox**.

OPTIONS

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, **mailx** attempts to read messages from the **mailbox**.

–**B**	Do not buffer standard input or standard output.
–**d**	Turn on debugging output. (Neither particularly interesting nor recommended.)
–**e**	Test for the presence of mail. **mailx** prints nothing and exits with a successful return code if there is mail to read.
–**F**	Record the message in a file named after the first recipient. Overrides the **record** variable, if set (see **Environment Variables**).
–**H**	Print header summary only.
–**i**	Ignore interrupts. See also **ignore** in **Environment Variables**.

–I	Include the newsgroup and article-id header lines when printing mail messages. This option requires the **–f** option to be specified.
–n	Do not initialize from the system default **mailx.rc** or **Mail.rc** file.
–N	Do not print initial header summary.
–U	Convert **uucp** style addresses to internet standards. Overrides the **conv** environment variable.
–v	Pass the **–v** flag to **sendmail**(1M).
–V	Print the **mailx** version number and exit.
–b *bcc*	Set the blind carbon copy list to *bcc*. *bcc* should be enclosed in quotes if it contains more than one name.
–c *cc*	Set the carbon copy list to *cc*. *cc* should be enclosed in quotes if it contains more than one name.
–f [*filename***]**	Read messages from *filename* instead of **mailbox**. If no *filename* is specified, the **mbox** is used.
–f [+*folder***]**	Use the file *folder* in the folder directory (same as the **fold**er command). The name of this directory is listed in the **folder** variable.
–h *number*	The number of network ‘‘hops’’ made so far. This is provided for network software to avoid infinite delivery loops. This option and its argument are passed to the delivery program.
–r *address*	Use *address* as the return address when invoking the delivery program. All tilde commands are disabled. This option and its argument is passed to the delivery program.
–s *subject*	Set the Subject header field to *subject*. *subject* should be enclosed in quotes if it contains embedded white space.
–t	Scan the input for ‘‘To:’’, ‘‘Cc:’’, and ‘‘Bcc:’’ fields. Any recipients on the command line will be ignored.
–T *file*	Message-id and article-id header lines are recorded in *file* after the message is read. This option also sets the **–I** option.
–u *user*	Read *user*’s **mailbox**. This is only effective if *user*’s **mailbox** is not read protected.
–~	Interpret tilde escapes in the input even if not reading from a tty.

USAGE

Starting Mail

At start-up time, **mailx** tries to execute commands from the optional system-wide file (**/etc/mail/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. If invoked as **mail** or **Mail**, the system-wide start-up file **/etc/mail/Mail.rc** is used instead. With the exceptions noted below, regular commands are legal inside start-up files.

The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **C**opy, **e**dit, **fo**llowup, **F**ollowup, **ho**ld, **m**ail, **pre**serve, **r**eply, **R**eply, **sh**ell, and **v**isual. An error in the start-up file causes the remaining lines in the file to be ignored. The **mailrc** file is optional, and must be constructed locally.

At any time, the behavior of **mailx** is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared using the **se**t and **uns**et commands. See **Environment Variables** below for a summary of these parameters.

When reading mail, **mailx** is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating **mailx** can accept regular commands (see **Commands** below). When sending mail, **mailx** is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed.

As the message is typed, **mailx** reads the message and stores it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See **Tilde Escapes** for a summary of these commands.

Reading Mail

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*message-list*) to operate on.

The default for *message-list* is the current message. A *message-list* is a list of message identifiers separated by spaces, which may include:

n	Message number *n*.
.	The current message.
^	The first undeleted message.
$	The last message.
*	All messages.
+	The next undeleted message.
–	The previous undeleted message.
n−*m*	An inclusive range of message numbers.
user	All messages from *user*.
/string	All messages with *string* in the Subject line (case ignored).
:*c*	All messages of type *c*, where *c* is one of: **d** deleted messages **n** new messages **o** old messages **r** read messages **u** unread messages Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. Filenames, where expected, are expanded using the normal shell conventions (see **sh**(1)). Special characters are recognized by certain commands and are documented with the commands below.

Sending Mail

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to be undeliverable, an attempt is made to return it to the sender's **mailbox**. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as **lp**(1) for recording outgoing mail on paper. Alias groups are set by the **a**lias command (see **Commands** below) or in a system start-up file (for example, **$HOME/.mailrc**). Aliases are lists of recipients of any type.

Forwarding Mail

To forward a specific message, include it in a message to the desired recipients with the **˜f** or **˜m** tilde escapes. See **Tilde Escapes** below. To forward mail automatically, add a comma-separated list of addresses for additional recipients to the **.forward** file in your home directory. This is different from the format of the **alias** command, which takes a space-separated list instead. Note: forwarding addresses must be valid, or the messages will "bounce." You cannot, for instance, reroute your mail to a new host by forwarding it to your new address if it is not yet listed in the NIS aliases domain.

Commands

Regular commands are of the form

[*command*] [*message-list*] [*arguments*]

In *input mode*, commands are recognized by the escape character, **˜**, and lines not treated as commands are taken as input for the message.

If no command is specified in *command mode*, **n**ext is assumed.

The following is a complete list of **mailx** commands:

!*shell-command*

Escape to the shell. See **SHELL** in **Environment Variables**.

*comment*

Null command (comment). Useful in **mailrc** files.

= Print the current message number.

? Prints a summary of commands.

alias *alias name* . . .
group *alias name* . . .

Declare an alias for the given names. The names are substituted when *alias* is used as a recipient. Useful in the **mailrc** file. With no arguments, the command displays the list of defined aliases.

alternates *name* . . .

Declare a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, print the current list of alternate names. See also **allnet** in **Environment Variables**.

cd [*directory*]
chdir [*directory*]

Change directory. If *directory* is not specified, **$HOME** is used.

copy [*filename*]
copy [*message-list*] *filename*

Copy messages to the file without marking the messages as saved. Otherwise equivalent to the **s**ave command.

Copy [*message-list*]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the **S**ave command.

delete [*message-list*]

Delete messages from the **mailbox**. If **autoprint** is set, the next message after the last one deleted is printed (see **Environment Variables**).

discard [*header-field*. . .]
ignore [*header-field*. . .]

Suppress printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are **Status** and **Received**. The fields are included when the message is saved, unless the **alwaysignore** variable is set. The **Mo**re, **Pa**ge, **P**rint, and **T**ype commands override this command. If no header is specified, the current list of header fields being ignored is printed. See also the **undi**scard and **unig**nore commands.

dp [*message-list*]
dt [*message-list*]

Delete the specified messages from the **mailbox** and print the next message after the last one deleted. Roughly equivalent to a **d**elete command followed by a **p**rint command.

echo *string* . . .

Echo the given strings (like **echo**(1)).

edit [*message-list*]

Edit the given messages. The messages are placed in a temporary file and the **EDITOR** variable is used to get the name of the editor (see **Environment Variables**). Default editor is **ed**(1).

exit
xit

Exit from **mailx**, without changing the **mailbox**. No messages are saved in the **mbox** (see also **q**uit).

field [*message-list*] header-file

Display the value of the header field in the specified message.

file [*filename*]
folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names:

%	the current **mailbox**.
%*user*	the **mailbox** for *user*.
#	the previous mail file.

&	the current **mbox**.
+*filename*	The named file in the *folder* directory (listed in the **folder** variable).

With no arguments, print the name of the current mail file, and the number of messages and characters it contains.

folders Print the names of the files in the directory set by the **folder** variable (see **Environment Variables**).

Followup [*message*]
Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the **record** variable, if set. See also the **f**ollowup, **S**ave, and **C**opy commands and **outfolder** in **Environment Variables**.

followup [*message-list*]
Respond to the first message in the *message-list*, sending the message to the author of each message in the *message-list*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **Fo**llowup, **S**ave, and **C**opy commands and **outfolder** in **Environment Variables**.

from [*message-list*]
Print the header summary for the specified messages. If no messages are specified, print the header summary for the current message.

group *alias name . . .*
alias *alias name . . .*
Declare an alias for the given names. The names are substituted when *alias* is used as a recipient. Useful in the **mailrc** file.

hheaders [*message*]
Print the page of headers which includes the message specified. The **screen** variable sets the number of headers per page (see **Environment Variables**). See also the **z** command.

help Print a summary of commands.

hold [*message-list*]
preserve [*message-list*]
Hold the specified messages in the **mailbox**.

if **s** | **r** | **t**
mail-commands
else
mail-commands
endif Conditional execution, where *s* executes following *mail-commands*, up to an **el**se or **en**dif, if the program is in *send* mode, *r* causes the *mail-commands* to be executed only in *receive* mode, and **t** causes the *mail-commands* to be executed only if **mailx** is being run from a terminal. Useful in the **mailrc** file.

inc Incorporate messages that arrive while you are reading the system mailbox. The new messages are added to the message list in the current **mail** session. This command does not commit changes made during the session, and prior messages are not renumbered.

ignore [*header-field* . . .]
discard [*header-field* . . .]

Suppress printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are **Status** and .BR Cc . All fields are included when the message is saved. The **Mo**re, **Pa**ge, **P**rint and **T**ype commands override this command. If no header is specified, the current list of header fields being ignored is printed. See also the **undi**scard and **unig**nore commands.

list Print all commands available. No explanation is given.

load [*message*] *filename*

The specified message is replaced by the message in the named file. *filename* should contain a single mail message including mail headers (as saved by the **s**ave command).

mail *recipient* . . .

Mail a message to the specified recipients.

Mail *recipient*

Mail a message to the specified recipients, and record it in a file whose name is derived from the auther of the message. Overrides the **record** variable, if set. See also the **Save** and **Copy** commands and **outfolder** in **Environment Variables**.

mbox [*message-list*]

Arrange for the given messages to end up in the standard **mbox** save file when **mailx** terminates normally. See **MBOX** in **Environment Variables** for a description of this file. See also the **ex**it and **q**uit commands.

more [*message-list*]
page [*message-list*]

Print the specified messages. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable are paged through the command specified by the **PAGER** variable. The default command is **pg**(1) or if the **bsdcompat** variable is set, the default is **more**(1). See **Environment Variables**. Same as the **p**rint and **t**ype commands.

More [*message-list*]
Page [*message-list*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command. Same as the **P**rint and **T**ype commands.

new [*message-list*]
New [*message-list*]
unread [*message-list*]
Unread [*message-list*]

Take a message list and mark each message as *not* having been read.

next [*message*]

Go to the next message matching *message*. If message is not supplied, this command finds the next message that was not deleted or saved. A *message-list* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *message-list* above for a description of possible message specifications.

pipe [*message-list*] [*shell-command*]
| [*message-list*] [*shell-command*]

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the **cmd** variable. If the **page** variable is set, a form feed character is inserted after each message (see **Environment Variables**).

preserve [*message-list*]
hold [*message-list*]

Preserve the specified messages in the **mailbox**.

print [*message-list*]
type [*message-list*]

Print the specified messages. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable are paged through the command specified by the **PAGER** variable. The default command is **pg**(1) or if the **bsdcompat** variable is set, the default is **more**(1). See **Environment Variables**. Same as the **mo**re and **pa**ge commands.

Print [*message-list*]
Type [*message-list*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command. Same as the **Mo**re and **Pa**ge commands.

put [*filename*]
put [*message-list*] *filename*

Save the specified message in the given filename. Use the same conventions as the **p**rint command for which header fields are ignored.

Put [*filename*]
Put [*message-list*] *filename*

Save the specified message in the given filename. Overrides suppression of fields by the **ig**nore command.

quit Exit from **mailx**, storing messages that were read in **mbox** and unread messages in the **mailbox**. Messages that have been explicitly saved in a file are deleted unless the **keepsave** variable is set.

reply [*message-list*]

respond [*message-list*]
replysender [*message-list*]

Send a response to the author of each message in the *message-list*. The subject line is taken from the first message. If **record** is set to a filename, a copy of the reply is added to that file. If the **replyall** variable is set, the actions of **R**eply/**R**espond and **r**eply/**r**espond are reversed. The **replys**ender command is not affected by the **replyall** variable, but sends each reply only to the sender of each message.

Reply [*message*]
Respond [*message*]
replyall [*message*]

Reply to the specified message, including all other recipients of that message. If the variable **record** is set to a filename, a copy of the reply added to that file. If the **replyall** variable is set, the actions of **R**eply/**R**espond and **r**eply/**r**espond are reversed. The **replya**ll command is not affected by the **replyall** variable, but always sends the reply to all recipients of the message.

retain Add the list of header fields named to the *retained list*. Only the header fields in the retain list are shown on your terminal when you print a message. All other header fields are suppressed. The set of retained fields specified by the **ret**ain command overrides any list of ignored fields specified by the **ig**nore command. The **T**ype and **P**rint commands can be used to print a message in its entirety. If **ret**ain is executed with no arguments, it lists the current set of retained fields.

Save [*message-list*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the **C**opy, **fo**llowup, and **F**ollowup commands and **outfolder** in **Environment Variables**.

save [*filename*]
save [*message-list*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The file defaults to **mbox**. The message is deleted from the **mailbox** when **mailx** terminates unless **keepsave** is set (see also **Environment Variables** and the **ex**it and **q**uit commands).

set
set *variable*
set *variable*=*string*
set *variable*=*number*

Define a *variable*. To assign a *value* to *variable*, separate the variable name from the value by an '=' (there must be no space before or after the '='). A variable may be given a null, string, or numeric *value*. To embed SPACE characters within a *value* enclose it in quotes.

With no arguments, **se**t displays all defined variables and any values they might have. See **Environment Variables** for a description of all predefined **mail** variables.

shell Invoke an interactive shell. See also **SHELL** in **Environment Variables**.

size [*message-list*]
Print the size in characters of the specified messages.

source *filename*
Read commands from the given file and return to command mode.

top [*message-list*]
Print the top few lines of the specified messages. If the **toplines** variable is set, it is taken as the number of lines to print (see **Environment Variables**). The default is 5.

touch [*message-list*]
Touch the specified messages. If any message in *message-list* is not specifically saved in a file, it is placed in the **mbox**, or the file specified in the **MBOX** environment variable, upon normal termination. See **ex**it and **q**uit.

Type [*message-list*]
Print [*message-list*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

type [*message-list*]
print [*message-list*]
Print the specified messages. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable are paged through the command specified by the **PAGER** variable. The default command is **pg**(1) See **Environment Variables**.

unalias [*alias*] . . .
ungroup [*alias*] . . .
Remove the definitions of the specified aliases.

undelete [*message-list*]
Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If **autoprint** is set, the last message of those restored is printed (see **Environment Variables**).

undiscard [*header-field* . . .]
unignore [*header-field* . . .]
Remove the specified header fields from the list being ignored. If no header fields are specified, all header fields are removed from the list being ignored.

unretain [*header-field* . . .]
Remove the specified header fields from the list being retained. If no header fields are specified, all header fields are removed from the list being retained.

unread [*message-list*]
Unread [*message-list*]

Same as the **ne**w command.

unset *variable . . .*

Erase the specified variables. If the variable was imported from the environment (that is, an environment variable or exported shell variable), it cannot be unset from within **mailx**.

version Print the current version and release date of the **mailx** utility.

visual [*message-list*]

Edit the given messages with a screen editor. The messages are placed in a temporary file and the **VISUAL** variable is used to get the name of the editor (see **Environment Variables**).

write [*message-list*] *filename*

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the **s**ave command.

xit
exit Exit from **mailx**, without changing the **mailbox**. No messages are saved in the **mbox** (see also **q**uit).

z[+ | −] Scroll the header display forward or backward one screen–full. The number of headers displayed is set by the **screen** variable (see **Environment Variables**).

Tilde Escapes

The following *tilde escape* commands can be used when composing mail to send. These may be entered only from *input mode*, by beginning a line with the tilde escape character (˜). See **escape** in **Environment Variables** for changing this special character. The escape character can be entered as text by typing it twice.

˜!*shell-command*

Escape to the shell. If present, run *shell-command*.

˜. Simulate end of file (terminate message input).

˜:*mail-command*
˜_ *mail-command*

Perform the command-level request. Valid only when sending a message while reading mail.

˜? Print a summary of tilde escapes.

˜A Insert the autograph string **Sign** into the message (see **Environment Variables**).

˜a Insert the autograph string **sign** into the message (see **Environment Variables**).

˜b *name . . .*

Add the *name*s to the blind carbon copy (**Bcc**) list. This is like the carbon copy (**Cc**) list, except that the names in the **Bcc** list are not shown in the header of the mail message.

~c *name* . . .
Add the *name*s to the carbon copy (Cc) list.

~d Read in the **dead-letter** file. See **DEAD** in **Environment Variables** for a description of this file.

~e Invoke the editor on the partial message. See also **EDITOR** in **Environment Variables**.

~f [*message-list*]
Forward the specified message, or the current message being read. Valid only when sending a message while reading mail. The messages are inserted into the message without alteration (as opposed to the **~m** escape).

~F [*message-list*]
Forward the specified message, or the current message being read, including all header fields. Overrides the suppression of fields by the **ig**nore command.

~h Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *variable*
Insert the value of the named variable into the text of the message. For example, **~A** is equivalent to '**~i Sign**.' Environment variables set and exported in the shell are also accessible by **~i**.

~m [*message-list*]
Insert the listed messages, or the current message being read into the letter. Valid only when sending a message while reading mail. The text of the message is shifted to the right, and the string contained in the **indentprefix** variable is inserted as the leftmost characters of each line. If **indentprefix** is not set, a TAB character is inserted into each line.

~M [*message-list*]
Insert the listed messages, or the current message being read, including the header fields, into the letter. Valid only when sending a message while reading mail. The text of the message is shifted to the right, and the string contained in the **indentprefix** variable is inserted as the leftmost characters of each line. If **indentprefix** is not set, a TAB character is inserted into each line. Overrides the suppression of fields by the **ig**nore command.

~p Print the message being entered.

~q Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in **dead-letter**. See **DEAD** in **Environment Variables** for a description of this file.

~r *filename*
~< *filename*
~< ! *shell-command*
Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

˜s *string ...*
Set the subject line to *string*.

˜t *name ...*
Add the given *name*s to the To list.

˜v Invoke a preferred screen editor on the partial message. The default visual editor is **vi**(1). See also **VISUAL** in **Environment Variables**.

˜w *filename*
Write the message into the given file, without the header.

˜x Exit as with **˜q** except the message is not saved in **dead-letter**.

˜| *shell-command*
Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

Environment Variables

The following are environment variables taken from the execution environment and are not alterable within **mailx**.

HOME=*directory*
The user's home directory.

MAIL=*filename*
The name of the initial mailbox file to read (in lieu of the standard system mailbox). The default is **/var/mail/***username*.

MAILRC=*filename*
The name of the start-up file. Default is **$HOME/.mailrc**.

The following variables are internal **mailx** variables. They may be imported from the execution environment or set using the **se**t command at any time. The **uns**et command may be used to erase variables.

allnet All network names whose last component (login name) match are treated as identical. This causes the *message-list* message specifications to behave similarly. Disabled by default. See also the **alt**ernates command and the **metoo** variable.

alwaysignore
Ignore header fields with **ig**nore everywhere, not just during **p**rint or **t**ype. Affects the **s**ave, **S**ave, **c**opy, **C**opy, **to**p, **pi**pe, and **w**rite commands, and the **˜m** and **˜f** tilde escapes. Enabled by default.

append
Upon termination, append messages to the end of the **mbox** file instead of prepending them. Disabled by default, but **append** is set in the global start-up file (which can be suppressed with the **–n** command line option).

askbcc Prompt for the Bcc list after the Subject is entered if it is not specified on the command line with the **–b** option. Disabled by default.

askcc Prompt for the Cc list after the Subject is entered if it is not specified on the command line with the **–c** option. Disabled by default.

asksub Prompt for subject if it is not specified on the command line with the −**s** option. Enabled by default.

autoinc
Automatically incorporate new messages into the current session as they arrive. This has an affect similar to issuing the **inc** command every time the command prompt is displayed. Disabled by default, but **autoinc** is set in the default system start-up file for **mailx**; it is not set for **/usr/ucb/mail** or **/usr/ucb/Mail**.

autoprint
Enable automatic printing of messages after **d**elete and **u**ndelete commands. Disabled by default.

bang Enable the special-casing of exclamation points (!) in shell escape command lines as in **vi**(1). Disabled by default.

bsdcompat
Set automatically if **mailx** is invoked as **mail** or **Mail**. Causes **mailx** to use **/etc/mail/Mail.rc** as the system start-up file. Changes the default pager to **more**(1).

cmd=*shell-command*
Set the default command for the **pi**pe command. No default value.

conv=*conversion*
Convert **uucp** addresses to the specified address style, which can be either:

internet
This requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing.

optimize
Remove loops in **uucp**(1C) address paths (typically generated by the **r**eply command). No rerouting is performed; **mail** has no knowledge of UUCP routes or connections.

Conversion is disabled by default. See also **sendmail**(1M) and the −**U** command-line option.

crt[**=***number*]
Pipe messages having more than *number* lines through the command specified by the value of the **PAGER** variable (**pg**(1) or **more**(1) by default). If *number* is not specified, the current window size is used. Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt. Default is **$HOME/dead.letter**.

debug Enable verbose diagnostics for debugging. Messages are not delivered. Disabled by default.

dot Take a period on a line by itself, or EOF during input from a terminal as end-of-file. Disabled by default, but **dot** is set in the global start-up file (which can be suppressed with the −**n** command line option).

EDITOR=*shell-command*
The command to run when the **e**dit or **˜e** command is used. Default is **ed**(1).

escape=*c*
Substitute *c* for the ˜ escape character. Takes effect with next message sent.

folder=*directory*
The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), **$HOME** is prepended to it. There is no default for the **folder** variable. See also **outfolder** below.

header Enable printing of the header summary when entering **mailx**. Enabled by default.

hold Preserve all messages that are read in the **mailbox** instead of putting them in the standard **mbox** save file. Disabled by default.

ignore Ignore interrupts while entering messages. Handy for noisy dial-up lines. Disabled by default.

ignoreeof
Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the **˜.** command. See also **dot** above. Disabled by default.

indentprefix=*string*
When **indentprefix** is set, *string* is used to mark indented lines from messages included with **˜m**. The default is a TAB character.

keep When the **mailbox** is empty, truncate it to zero length instead of removing it. Disabled by default.

iprompt=*string*
The specified prompt string is displayed before each line on input is requested when sending a message.

keepsave
Keep messages that have been saved in other files in the **mailbox** instead of deleting them. Disabled by default.

LISTER=*shell-command*
The command (and options) to use when listing the contents of the **folder** directory. The default is **ls**(1).

MAILX_HEAD=*string*
The specified string is included at the beginning of the body of each message that is sent.

MAILX_TAIL=*string*
The specified string is included at the end of the body of each message that is sent.

makeremote

When replying to all recipients of a message, if an address does not include a machine name, it is assumed to be relative to the sender of the message. Normally not needed when dealing with hosts that support RFC822.

MBOX=*filename*
The name of the file to save messages which have been read. The **ex**it command overrides this function, as does saving the message explicitly in another file. Default is **$HOME/mbox**.

metoo If your login appears as a recipient, do not delete it from the list. Disabled by default.

mustbang
Force all mail addresses to be in bang format.

onehop
When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away). Disabled by default.

outfolder
Locate the files used to record outgoing messages in the directory specified by the **folder** variable unless the path name is absolute. Disabled by default. See **folder** above and the **S**ave, **C**opy, **fo**llowup, and **F**ollowup commands.

page Used with the **pi**pe command to insert a form feed after each message sent through the pipe. Disabled by default.

PAGER=*shell-command*
The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is **pg**(1), or if the **bsdcompat** variable is set, the default is **more**(1). See **Environment Variables**.

postmark
Your "real name" to be included in the From line of messages you send. By default this is derived from the comment field in your passwd file entry.

prompt=*string*
Set the *command mode* prompt to *string*. Default is ‘‘**?** ’’, unless the **bsdcompat** variable is set, then the default is ‘‘**&**’’.

quiet Refrain from printing the opening message and version when entering **mailx**. Disabled by default.

record=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also **outfolder** above. If you have the **record** and **outfolder** variables set but the **folder** variable not set, messages are saved in **+***filename* instead of *filename*.

replyall
Reverse the effect of the **r**eply and **R**eply commands.

save Enable saving of messages in **dead-letter** on interrupt or delivery error. See **DEAD** for a description of this file. Enabled by default.

screen=*number*
Sets the number of lines in a screen-full of headers for the **h**headers command. *number* must be a positive number.

The default is set according to baud rate or window size. With a baud rate less than 1200, *number* defaults to 5, if baud rate is exactly 1200, it defaults to 10. If you are in a window, *number* defaults to the default window size minus 4. Otherwise, the default is 20.

sendmail=*shell-command*
Alternate command for delivering messages. Note: in addition to the expected list of recipients, **mail** also passes the –**i** and –**m**, flags to the command. Since these flags are not appropriate to other commands, you may have to use a shell script that strips them from the arguments list before invoking the desired command. Default is **/usr/bin/rmail**.

sendwait
Wait for background mailer to finish before returning. Disabled by default.

SHELL=*shell-command*
The name of a preferred command interpreter. Default is **sh**(1).

showname
Causes the message header display to show the sender's real name (if known) rather than their mail address. Disabled by default, but **showname** is set in the **/etc/mail/mailx.rc** system start-up file for **mailx**.

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
The variable inserted into the text of a message when the **˜a** (autograph) command is given. No default (see also **˜i** in **Tilde Escapes**).

Sign=*string*
The variable inserted into the text of a message when the **˜A** command is given. No default (see also **˜i** in **Tilde Escapes**).

toplines=*number*
The number of lines of header to print with the **to**p command. Default is 5.

verbose
Invoke **sendmail**(1M) with the –**v** flag.

translate

The name of a program to translate mail addresses. The program receives mail addresses as arguments. The program produces, on the standard output, lines containing the following data, in this order:

- the postmark for the sender (see the postmark variable)
- translated mail addresses, one per line, corresponding to the program's arguments. Each translated address will replace the corresponding address in the mail message being sent.
- a line containing only "y" or "n". if the line contains "y" the user will be asked to confirm that the message should be sent.

The translate program will be invoked for each mail message to be sent. If the program exits with a non-zero exit status, or fails to produce enough output, the message is not sent.

VISUAL=*shell-command*

The name of a preferred screen editor. Default is **vi**(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/etc/mail/mailx.rc	optional global start-up file for **mailx** only
/etc/mail/Mail.rc	BSD compatibility system-wide start-up file for **/usr/ucb/mail** and **/usr/ucb/Mail**
/tmp/R[emqsx]∗	temporary files
/usr/share/lib/mailx/mailx.help∗	help message files
/var/mail/∗	post office directory

SEE ALSO

biff(1B), **echo**(1), **ed**(1), **ex**(1), **fmt**(1), **ls**(1), **mail**(1), **more**(1), **newaliases**(1), **pg**(1), **sh**(1), **vacation**(1), **vi**(1), **uucp**(1C), **sendmail**(1M), **aliases**(4)

NOTES

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **uns**et.

The full internet addressing is not fully supported by **mailx**. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a ‘‘.’’ are treated as the end of the message by **mail**(1) (the standard mail delivery program).

Replies do not always generate correct return addresses. Try resending the errant reply with **onehop** set.

mailx does not lock your record file. So, if you use a record file and send two or more messages simultaneously, lines from the messages may be interleaved in the record file.

The format for the **alias** command is a space-separated list of recipients, while the format for an alias in either the **.forward** or **/etc/aliases** is a comma-separated list.

NAME

make – maintain, update, and regenerate related programs and files

SYNOPSIS

/usr/ccs/bin/make [**−f** *makefile*] . . . [**−d**] [**−dd**] [**−D**] [**−DD**] [**−e**] [**−i**] [**−k**]
[**−n**] [**−p**] [**−P**] [**−q**] [**−r**] [**−s**] [**−S**] [**−t**] [*target* . . .] [*macro***=***value* . . .]

DESCRIPTION

make executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries that describe how to bring a target up to date with respect to those on which it depends, which are called *dependencies*. Since each dependency is a target, it may have dependencies of its own. Targets, dependencies, and sub-dependencies comprise a tree structure that **make** traces when deciding whether or not to rebuild a *target*.

make recursively checks each *target* against its dependencies, beginning with the first target entry in *makefile* if no *target* argument is supplied on the command line. If, after processing all of its dependencies, a target file is found either to be missing, or to be older than any of its dependencies, **make** rebuilds it. Optionally with this version of **make**, a target can be treated as out-of-date when the commands used to generate it have changed since the last time the target was built.

To build a given target, **make** executes the list of commands, called a *rule*. This rule may be listed explicitly in the target's makefile entry, or it may be supplied implicitly by **make**.

When no *makefile* is specified with a **−f** option:

- If there is a file named **makefile** in the working directory, **make** uses that file. If, however, there is an SCCS history file (**SCCS/s.makefile**) which is newer, **make** attempts to retrieve and use the most recent version.
- In the absence of the above file(s), if a file named **Makefile** is present in the working directory, **make** attempts to use it. If there is an SCCS history file (**SCCS/s.Makefile**) that is newer, **make** attempts to retrieve and use the most recent version.

If no *target* is specified on the command line, **make** uses the first target defined in *makefile*.

If a *target* has no makefile entry, or if its entry has no rule, **make** attempts to derive a rule by each of the following methods, in turn, until a suitable rule is found. (Each method is described under **USAGE** below.)

- Pattern matching rules.
- Implicit rules, read in from a user-supplied makefile.
- Standard implicit rules (also known as suffix rules), typically read in from the file **</usr/share/lib/make/make.rules>**.
- SCCS retrieval. **make** retrieves the most recent version from the SCCS history file (if any). See the description of the **.SCCS_GET:** special-function target for details.

- The rule from the **.DEFAULT:** target entry, if there is such an entry in the makefile.

If there is no makefile entry for a *target*, if no rule can be derived for building it, and if no file by that name is present, **make** issues an error message and halts.

OPTIONS

–f *makefile*
Use the description file *makefile*. A '–' as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the standard set of implicit rules and predefined macros. When more than one '**–f** *makefile*' argument pair appears, **make** uses the concatenation of those files, in order of appearance.

–d Display the reasons why **make** chooses to rebuild a target; **make** displays any and all dependencies that are newer. In addition, **make** displays options read in from the **MAKEFLAGS** environment variable.

–dd Display the dependency check and processing in vast detail.

–D Display the text of the makefiles read in.

–DD Display the text of the makefiles, **make.rules** file, the state file, and all hidden-dependency reports.

–e Environment variables override assignments within makefiles.

–i Ignore error codes returned by commands. Equivalent to the special-function target '**.IGNORE:**'.

–k When a nonzero error status is returned by a rule, or when **make** cannot find a rule, abandon work on the current target, but continue with other dependency branches that do not depend on it.

–n No execution mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed. However, if a command line contains a reference to the **$(MAKE)** macro, that line is always executed (see the discussion of **MAKEFLAGS** in **Reading Makefiles and the Environment**).

–p Print out the complete set of macro definitions and target descriptions.

–P Merely report dependencies, rather than building them.

–q Question mode. **make** returns a zero or nonzero status code depending on whether or not the target file is up to date.

–r Do not read in the default makefile **</usr/share/lib/make/make.rules>**.

–s Silent mode. Do not print command lines before executing them. Equivalent to the special-function target **.SILENT:**.

–S Undo the effect of the **–k** option. Stop processing when a non-zero exit status is returned by a command.

–t Touch the target files (bringing them up to date) rather than performing their rules. *This can be dangerous when files are maintained by more than one person.* When the **.KEEP_STATE:** target appears in the makefile, this option updates the state file just as if the rules had been performed.

macro=*value*
Macro definition. This definition overrides any regular definition for the specified macro within the makefile itself, or in the environment. However, this definition can still be overridden by conditional macro assignments.

USAGE

Refer to **make** in *Programming Utilities Guide* for tutorial information.

Reading Makefiles and the Environment

When **make** first starts, it reads the **MAKEFLAGS** environment variable to obtain any the following options specified present in its value: −**d**, −**D**, −**e**, −**i**, −**k**, −**l**, −**n**, −**p**, −**q**, −**r**, −**s**, −**S**, or −**t**. (Within the **MAKEFLAGS** value, the leading '**—**' character for the option string is omitted.) **make** then reads the command line for additional options, which also take effect.

Next, **make** reads in a default makefile that typically contains predefined macro definitions, target entries for implicit rules, and additional rules, such as the rule for retrieving SCCS files. If present, **make** uses the file **make.rules** in the current directory; otherwise it reads the file **</usr/share/lib/make/make.rules>**, which contains the standard definitions and rules.
Use the directive:

include </usr/share/lib/make/make.rules>

in your local **make.rules** file to include them.

Next, **make** imports variables from the environment (unless the −**e** option is in effect), and treats them as defined macros. Because **make** uses the most recent definition it encounters, a macro definition in the makefile normally overrides an environment variable of the same name. When −**e** is in effect, however, environment variables are read in *after* all makefiles have been read. In that case, the environment variables take precedence over definitions in the makefile.

Next, **make** reads the state file, **.make.state** in the local directory if it exists, and then any makefiles you specify with −**f**, or one of **makefile** or **Makefile** as described above.

Next, (after reading the environment if −**e** is in effect), **make** reads in any macro definitions supplied as command line arguments. These override macro definitions in the makefile and the environment both, but only for the **make** command itself.

make exports environment variables, using the most recently defined value. Macro definitions supplied on the command line are not normally exported, unless the macro is also an environment variable.

make does not export macros defined in the makefile. If an environment variable is set, and a macro with the same name is defined on the command line, **make** exports its value as defined on the command line. Unless −**e** is in effect, macro definitions within the makefile take precedence over those imported from the environment.

The macros **MAKEFLAGS**, **MAKE**, **SHELL**, **HOST_ARCH**, **HOST_MACH**, and **TARGET_MACH** are special cases. See **Special-Purpose Macros**, below, for details.

Makefile Target Entries

A target entry has the following format:

target. . . [**:** | **::**] [*dependency*] . . . [**;** *command*] . . .
[*command*]
. . .

The first line contains the name of a target, or a space-separated list of target names, terminated with a colon or double colon. If a list of targets is given, this is equivalent to having a separate entry of the same form for each target. The colon(s) may be followed by a *dependency*, or a dependency list. **make** checks this list before building the target. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a single Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise the rule for building the target.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, **make** expands macros, strips off initial TAB characters and either executes the command directly (if it contains no shell metacharacters), or passes each command line to a Bourne shell for execution.

The first line that does not begin with a TAB or **#** begins another target or macro definition.

Special Characters

Global

Start a comment. The comment ends at the next NEWLINE. If the **#** follows the TAB in a command line, that line is passed to the shell (which also treats **#** as the start of a comment).

include *filename*
If the word **include** appears as the first seven letters of a line and is followed by a SPACE or TAB, the string that follows is taken as a filename to interpolate at that line. **include** files can be nested to a depth of no more than about 16. If *filename* is a macro reference, it is expanded.

Targets and Dependencies

: Target list terminator. Words following the colon are added to the dependency list for the target or targets. If a target is named in more than one colon-terminated target entry, the dependencies for all its entries are added to form that target's complete dependency list.

:: Target terminator for alternate dependencies. When used in place of a '**:**' the double-colon allows a target to be checked and updated with respect to alternate dependency lists. When the target is out-of-date with respect to dependencies listed in the first alternate, it is built according to the rule for that entry. When out-of-date with respect to dependencies in another alternate, it is built according the rule in that other entry. Implicit rules do not apply to double-colon targets; you must supply a rule for each entry. If no dependencies are specified, the rule is always performed.

target [+ *target*. . .] **:**
Target group. The rule in the target entry builds all the indicated targets as a group. It is normally performed only once per **make** run, but is checked for command dependencies every time a target in the group is encountered in the dependency scan.

% Pattern matching wild card metacharacter. Like the '*' shell wild card, '**%**' matches any string of zero or more characters in a target name or dependency, in the target portion of a conditional macro definition, or within a pattern replacement macro reference. Note: only one '**%**' can appear in a target, dependency-name, or pattern-replacement macro reference.

./pathname
make ignores the leading '**./**' characters from targets with names given as pathnames relative to "dot," the working directory.

Macros

= Macro definition. The word to the left of this character is the macro name; words to the right comprise its value. Leading and trailing white space characters are stripped from the value. A word break following the **=** is implied.

$ Macro reference. The following character, or the parenthesized or bracketed string, is interpreted as a macro reference: **make** expands the reference (including the **$**) by replacing it with the macro's value.

( )
{ } Macro-reference name delimiters. A parenthesized or bracketed word appended to a **$** is taken as the name of the macro being referred to. Without the delimiters, **make** recognizes only the first character as the macro name.

$$** A reference to the dollar-sign macro, the value of which is the character '**$'. Used to pass variable expressions beginning with **$** to the shell, to refer to environment variables which are expanded by the shell, or to delay processing of dynamic macros within the dependency list of a target, until that target is actually processed.

\$ Escaped dollar-sign character. Interpreted as a literal dollar sign within a rule.

+= When used in place of '=', appends a string to a macro definition (must be surrounded by white space, unlike '=').

:= Conditional macro assignment. When preceded by a list of targets with explicit target entries, the macro definition that follows takes effect when processing only those targets, and their dependencies.

:sh = Define the value of a macro to be the output of a command (see **Command Substitutions**, below).

:sh In a macro reference, execute the command stored in the macro, and replace the reference with the output of that command (see **Command Substitutions**).

Rules

– **make** ignores any nonzero error code returned by a command line for which the first non-TAB character is a '–'. This character is not passed to the shell as part of the command line. **make** normally terminates when a command returns nonzero status, unless the –**i** or –**k** options, or the **.IGNORE:** special-function target is in effect.

@ If the first non-TAB character is a **@**, **make** does not print the command line before executing it. This character is not passed to the shell.

? Escape command-dependency checking. Command lines starting with this character are not subject to command dependency checking.

! Force command-dependency checking. Command-dependency checking is applied to command lines for which it would otherwise be suppressed. This checking is normally suppressed for lines that contain references to the '**?**' dynamic macro (for example, '**$?**').

When any combination of '–', '**@**', '**?**', or '**!**' appear as the first characters after the TAB, all that are present apply. None are passed to the shell.

Special-Function Targets

When incorporated in a makefile, the following target names perform special-functions:

.DEFAULT:
If it has an entry in the makefile, the rule for this target is used to process a target when there is no other entry for it, no rule for building it, and no SCCS history file from which to retrieve a current version. **make** ignores any dependencies for this target.

.DONE: If defined in the makefile, **make** processes this target and its dependencies after all other targets are built. This target is also performed when **make** halts with an error, unless the **.FAILED** target is defined.

.FAILED:
This target, along with its dependencies, is performed instead of **.DONE** when defined in the makefile and **make** halts with an error.

.IGNORE:
Ignore errors. When this target appears in the makefile, **make** ignores non-zero error codes returned from commands.

.INIT: If defined in the makefile, this target and its dependencies are built before any other targets are processed.

.KEEP_STATE:
If this target appears in the makefile, **make** updates the state file, **.make.state**, in the current directory. This target also activates command dependencies, and hidden dependency checks. If either the **.KEEP_STATE:** target appears in the makefile, or the environment variable **KEEP_STATE** is set ("setenv KEEP_STATE"), **make** will rebuild everything in order to collect dependency information, even if all the targets were up to date due to previous **make** runs. (See also the **ENVIRONMENT** section.)

.MAKE_VERSION:
A target-entry of the form:

.MAKE_VERSION: VERSION–*number*

enables version checking. If the version of **make** differs from the version indicated, **make** issues a warning message.

.NO_PARALLEL:
Currently, this target has no effect, it is, however, reserved for future use.

.PARALLEL:
Currently of no effect, but reserved for future use.

.PRECIOUS:
List of files not to delete. **make** does not remove any of the files listed as dependencies for this target when interrupted. **make** normally removes the current target when it receives an interrupt.

.SCCS_GET:
This target contains the rule for retrieving the current version of an SCCS file from its history file. To suppress automatic retrieval, add an entry for this target with an empty rule to your makefile.

.SILENT:
Run silently. When this target appears in the makefile, **make** does not echo commands before executing them.

.SUFFIXES:
The suffixes list for selecting implicit rules (see **The Suffixes List**).

.WAIT: Currently of no effect, but reserved for future use.

Clearing Special Targets

In this version of **make**, you can clear the definition of the following special targets by supplying entries for them with no dependencies and no rule:

.DEFAULT, **.SCCS_GET**, and **.SUFFIXES**

Command Dependencies

When the **.KEEP_STATE:** target appears in the makefile, **make** checks the command for building a target against the state file, **.make.state**. If the command has changed since the last **make** run, **make** rebuilds the target.

Hidden Dependencies

When the **.KEEP_STATE**: target appears in the makefile, **make** reads reports from **cpp**(1) and other compilation processors for any “hidden” files, such as **#include** files. If the target is out of date with respect to any of these files, **make** rebuilds it.

Macros

Entries of the form

macro=*value*

define macros. *macro* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. **make** strips both leading and trailing white space in accepting the value.

Subsequent references to the macro, of the forms: **$(**_name_**)** or **${**_name_**}** are replaced by _value_. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macro references can contain references to other macros, in which case nested references are expanded first.

Suffix Replacement Macro References

Substitutions within macros can be made as follows:

$(_name_**:**_string1_**=**_string2)_

where _string1_ is either a suffix, or a word to be replaced in the macro definition, and _string2_ is the replacement suffix or word. Words in a macro value are separated by SPACE, TAB, and escaped NEWLINE characters.

Pattern Replacement Macro References

Pattern matching replacements can also be applied to macros, with a reference of the form:

$(_name_**:** _op_**%**_os_**=** _np_**%**_ns_**)**

where _op_ is the existing (old) prefix and _os_ is the existing (old) suffix, _np_ and _ns_ are the new prefix and new suffix, respectively, and the pattern matched by **%** (a string of zero or more characters), is carried forward from the value being replaced. For example:

PROGRAM=fabricate
DEBUG= $(PROGRAM:%=tmp/%−g)

sets the value of **DEBUG** to **tmp/fabricate−g**.

Note: pattern replacement macro references cannot be used in the dependency list of a pattern matching rule; the **%** characters are not evaluated independently. Also, any number of **%** metacharacters can appear after the equal-sign.

Appending to a Macro

Words can be appended to macro values as follows:

macro **+=** _word_ . . .

Special-Purpose Macros

When the **MAKEFLAGS** variable is present in the environment, **make** takes options from it, in combination with options entered on the command line. **make** retains this combined value as the **MAKEFLAGS** macro, and exports it automatically to each command or shell it invokes.

Note: flags passed by way of **MAKEFLAGS** are only displayed when the **−d**, or **−dd** options are in effect.

The **MAKE** macro is another special case. It has the value **make** by default, and temporarily overrides the **−n** option for any line in which it is referred to. This allows nested invocations of **make** written as:

$(MAKE) . . .

to run recursively, with the **−n** flag in effect for all commands but **make.** This lets you use ‘**make −n**’ to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD **make**, the **MFLAGS** macro is set from the **MAKEFLAGS** variable by prepending a '–'. **MFLAGS** is not exported automatically.

The **SHELL** macro, when set to a single-word value such as **/usr/bin/csh**, indicates the name of an alternate shell to use. The default is **/bin/sh**. Note: **make** executes commands that contain no shell metacharacters itself. Built-in commands, such as **dirs** in the C shell, are not recognized unless the command line includes a metacharacter (for instance, a semicolon). This macro is neither imported from, nor exported to the environment, regardless of –**e**. To be sure it is set properly, you must define this macro within every makefile that requires it.

The following macros are provided for use with cross-compilation:

HOST_ARCH
The machine architecture of the host system. By default, this is the output of the **arch**(1B) command prepended with '—'. Under normal circumstances, this value should never be altered by the user.

HOST_MACH
The machine architecture of the host system. By default, this is the output of the **mach**(1B), prepended with '—'. Under normal circumstances, this value should never be altered by the user.

TARGET_ARCH
The machine architecture of the target system. By default, the output of **mach**, prepended with '—'.

Dynamic Macros

There are several dynamically maintained macros that are useful as abbreviations within rules. They are shown here as references; if you were to define them, **make** would simply override the definition.

$*	The basename of the current target, derived as if selected for use with an implicit rule.
$<	The name of a dependency file, derived as if selected for use with an implicit rule.
$@	The name of the current target. This is the only dynamic macro whose value is strictly determined when used in a dependency list. (In which case it takes the form '**$$@**'.)
$?	The list of dependencies that are newer than the target. Command-dependency checking is automatically suppressed for lines that contain this macro, just as if the command had been prefixed with a '**?**'. See the description of '**?**', under **Makefile Special Tokens**, above. You can force this check with the **!** command-line prefix.
$%	The name of the library member being processed. (See **Library Maintenance**, below.)

To refer to the **$@** dynamic macro within a dependency list, precede the reference with an additional ‘**$**’ character (as in, ‘**$$@**’). Because **make** assigns **$<** and **$**∗ as it would for implicit rules (according to the suffixes list and the directory contents), they may be unreliable when used within explicit target entries.

These macros can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case **F** or **D**, respectively (and enclosing the resulting name in parentheses or braces). Thus, ‘**$(@D)**’ refers to the directory part of the string ‘**$@**’; if there is no directory part, ‘**.**’ is assigned. **$(@F)** refers to the filename part.

Conditional Macro Definitions

A macro definition of the form:

target-list **:=** *macro* **=** *value*

indicates that when processing any of the targets listed *and their dependencies* , *macro* is to be set to the *value* supplied. Note that if a conditional macro is referred to in a dependency list, the **$** must be delayed (use **$$** instead). Also, *target-list* may contain a **%** pattern, in which case the macro will be conditionally defined for all targets encountered that match the pattern. A pattern replacement reference can be used within the *value*.

You can temporarily append to a macro’s value with a conditional definition of the form:

target-list **:=** *macro* **+=** *value*

Predefined Macros

make supplies the macros shown in the table that follows for compilers and their options, host architectures, and other commands. Unless these macros are read in as environment variables, their values are not exported by **make**. If you run **make** with any of these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each is expected to take. If –**r** is in effect, **make** does not read the default makefile (**./make.rules** or **</usr/share/lib/make/make.rules>**) in which these macro definitions are supplied.

Table of Predefined Macros

Use	*Macro*	*Default Value*
Library *Archives*	**AR**	**ar**
	ARFLAGS	**rv**
Assembler Commands	**AS**	**as**
	ASFLAGS	
	COMPILE.s	**$(AS) $(ASFLAGS)**
	COMPILE.S	**$(CC) $(ASFLAGS) $(CPPFLAGS) −c**
C Compiler Commands	**CC**	**cc**
	CFLAGS	
	CPPFLAGS	
	COMPILE.c	**$(CC) $(CFLAGS) $(CPPFLAGS) −c**
	LINK.c	**$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)**
C++ Compiler Commands	**CCC**	**CC**
	CCFLAGS	**CFLAGS**
	CPPFLAGS	
	COMPILE.cc	**$(CCC) $(CCFLAGS) $(CPPFLAGS) −c**
	LINK.cc	**$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)**
FORTRAN 77 Compiler Commands	**FC**	**f77**
	FFLAGS	
	COMPILE.f	**$(FC) $(FFLAGS) −c**
	LINK.f	**$(FC) $(FFLAGS) $(LDFLAGS)**
	COMPILE.F	**$(FC) $(FFLAGS) $(CPPFLAGS) −c**
	LINK.F	**$(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS)**
Link Editor Command	**LD**	**ld**
	LDFLAGS	
lex Command	**LEX**	**lex**
	LFLAGS	
	LEX.l	**$(LEX) $(LFLAGS) −t**
lint Command	**LINT**	**lint**
	LINTFLAGS	
	LINT.c	**$(LINT) $(LINTFLAGS) $(CPPFLAGS)**
Modula 2 Commands	**M2C**	**m2c**
	M2FLAGS	
	MODFLAGS	
	DEFFLAGS	
	COMPILE.def	**$(M2C) $(M2FLAGS) $(DEFFLAGS)**
	COMPILE.mod	**$(M2C) $(M2FLAGS) $(MODFLAGS)**
Pascal Compiler Commands	**PC**	**pc**
	PFLAGS	
	COMPILE.p	**$(PC) $(PFLAGS) $(CPPFLAGS) −c**
	LINK.p	**$(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS)**

Table of Predefined Macros

Use	*Macro*	*Default Value*
Ratfor	**RFLAGS**	
Compilation	**COMPILE.r**	**$(FC) $(FFLAGS) $(RFLAGS) −c**
Commands	**LINK.r**	**$(FC) $(FFLAGS) $(RFLAGS) $(LDFLAGS)**
rm **Command**	**RM**	**rm −f**
sccs	**SCCSFLAGS**	
Command	**SCCSGETFLAGS**	**−s**
yacc	**YACC**	**yacc**
Command	**YFLAGS**	
	YACC.y	**$(YACC) $(YFLAGS)**
Suffixes **List**	**SUFFIXES**	**.o .c .c˜ .cc .cc˜ .y .y˜ .l .l˜ .s .s˜ .sh .sh˜ .S .S˜ .ln .h .h˜ .f .f˜ .F .F˜ .mod .mod˜ .sym .def .def˜ .p .p˜ .r .r˜ .cps .cps˜ .C .C˜ .Y .Y˜ .L .L˜**

Implicit Rules

When a target has no entry in the makefile, **make** attempts to determine its class (if any) and apply the rule for that class. An implicit rule describes how to build any target of a given class, from an associated dependency file. The class of a target can be determined either by a pattern, or by a suffix; the corresponding dependency file (with the same basename) from which such a target might be built. In addition to a predefined set of implicit rules, make allows you to define your own, either by pattern, or by suffix.

Pattern Matching Rules

A target entry of the form:

tp%*ts*: *dp*%*ds*
rule

is a pattern matching rule, in which *tp* is a target prefix, *ts* is a target suffix, *dp* is a dependency prefix, and *ds* is a dependency suffix (any of which may be null). The **%** stands for a basename of zero or more characters that is matched in the target, and is used to construct the name of a dependency. When **make** encounters a match in its search for an implicit rule, it uses the rule in that target entry to build the target from the dependency file. Pattern-matching implicit rules typically make use of the **$@** and **$<** dynamic macros as placeholders for the target and dependency names. Other, regular dependencies may occur in the dependency list. An entry of the form:

tp %*ts*: [*dependency* . . .] *dp* %*ds* [*dependency* . . .]
rule

is a valid pattern matching rule.

Suffix Rules

When no pattern matching rule applies, **make** checks the target name to see if it ends with a suffix in the known suffixes list. If so, **make** checks for any suffix rules, as well as a dependency file with same root and another recognized suffix, from which to build it.

The target entry for a suffix rule takes the form:

DsTs: *rule*

where *Ts* is the suffix of the target, *Ds* is the suffix of the dependency file, and *rule* is the rule for building a target in the class. Both *Ds* and *Ts* must appear in the suffixes list. (A suffix need not begin with a '**.**' to be recognized.)

A suffix rule with only one suffix describes how to build a target having a null (or no) suffix from a dependency file with the indicated suffix. For instance, the **.c** rule could be used to build an executable program named **file** from a C source file named '**file.c**'. If a target with a null suffix has an explicit dependency, **make** omits the search for a suffix rule.

Table of Standard Implicit (Suffix) Rules

Use	*Implicit Rule Name*	*Command Line*
Assembly Files	**.s.o**	**$(COMPILE.s) −o $@ $<**
	.s.a	**$(COMPILE.s) −o $% $<** **$(AR) $(ARFLAGS) $@ $%** **$(RM) $%**
	.S.o	**$(COMPILE.S) −o $@ $<**
	.S.a	**$(COMPILE.S) −o $% $<** **$(AR) $(ARFLAGS) $@ $%** **$(RM) $%**
C **Files**	**.c**	**$(LINK.c) −o $@ $< $(LDLIBS)**
	.c.ln	**$(LINT.c) $(OUTPUT_OPTION) −i $<**
	.c.o	**$(COMPILE.c) $(OUTPUT_OPTION) $<**
	.c.a	**$(COMPILE.c) −o $% $<** **$(AR) $(ARFLAGS) $@ $%** **$(RM) $%**
C++ **Files**	**.cc**	**$(LINK.cc) −o $@ $< $(LDLIBS)**
	.cc.o	**$(COMPILE.cc) $(OUTPUT_OPTION) $<**
	.cc.a	**$(COMPILE.cc) −o $% $<** **$(AR) $(ARFLAGS) $@ $%** **$(RM) $%**
FORTRAN 77 **Files**	**.f**	**$(LINK.f) −o $@ $< $(LDLIBS)**
	.f.o	**$(COMPILE.f) $(OUTPUT_OPTION) $<**
	.f.a	**$(COMPILE.f) −o $% $<** **$(AR) $(ARFLAGS) $@ $%** **$(RM) $%**
	.F	**$(LINK.F) −o $@ $< $(LDLIBS)**
	.F.o	**$(COMPILE.F) $(OUTPUT_OPTION) $<**
	.F.a	**$(COMPILE.F) −o $% $<** **$(AR) $(ARFLAGS) $@ $%** **$(RM) $%**

Table of Standard Implicit (Suffix) Rules

Use	*Implicit Rule Name*	*Command Line*
lex *Files*	**.l**	**$(RM) $*.c** **$(LEX.l) $< > $*.c** **$(LINK.c) –o $@ $*.c $(LDLIBS)** **$(RM) $*.c**
	.l.c	**$(RM) $@** **$(LEX.l) $< > $@**
	.l.ln	**$(RM) $*.c** **$(LEX.l) $< > $*.c** **$(LINT.c) –o $@ –i $*.c** **$(RM) $*.c**
	.l.o	**$(RM) $*.c** **$(LEX.l) $< > $*.c** **$(COMPILE.c) –o $@ $*.c** **$(RM) $*.c**
Modula 2	**.mod**	**$(COMPILE.mod) –o $@ –e $@ $<**
Files	**.mod.o**	**$(COMPILE.mod) –o $@ $<**
	.def.sym	**$(COMPILE.def) –o $@ $<**
NeWS	**.cps.h**	**cps $*.cps**
Pascal	**.p**	**$(LINK.p) –o $@ $< $(LDLIBS)**
Files	**.p.o**	**$(COMPILE.p) $(OUTPUT_OPTION) $<**
Ratfor	**.r**	**$(LINK.r) –o $@ $< $(LDLIBS)**
Files	**.r.o**	**$(COMPILE.r) $(OUTPUT_OPTION) $<**
	.r.a	**$(COMPILE.r) –o $% $<** **$(AR) $(ARFLAGS) $@ $%** **$(RM) $%**
SCCS **Files**	**.SCCS_GET**	**sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@ –G$@**
Shell *Scripts*	**.sh**	**cat $< >$@** **chmod +x $@**
yacc *Files*	**.y**	**$(YACC.y) $<** **$(LINK.c) –o $@ y.tab.c $(LDLIBS)** **$(RM) y.tab.c**
	.y.c	**$(YACC.y) $<** **mv y.tab.c $@**
	.y.ln	**$(YACC.y) $<** **$(LINT.c) –o $@ –i y.tab.c** **$(RM) y.tab.c**
	.y.o	**$(YACC.y) $<** **$(COMPILE.c) –o $@ y.tab.c** **$(RM) y.tab.c**

make reads in the standard set of implicit rules from the file **</usr/share/lib/make/make.rules>**, unless **–r** is in effect, or there is a **make.rules** file in the local directory that does not **include** that file.

The Suffixes List

The suffixes list is given as the list of dependencies for the '**.SUFFIXES:**' special-function target. The default list is contained in the **SUFFIXES** macro (See *Table of Predefined Macros* for the standard list of suffixes). You can define additional **.SUFFIXES**: targets; a **.SUFFIXES** target with no dependencies clears the list of suffixes. Order is significant within the list; **make** selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list. To place suffixes at the head of the list, clear the list and replace it with the new suffixes, followed by the default list:

.SUFFIXES:
.SUFFIXES: *suffixes* **$(SUFFIXES)**

A tilde (˜) indicates that if a dependency file with the indicated suffix (minus the ˜) is under SCCS its most recent version should be retrieved, if necessary, before the target is processed.

Library Maintenance

A target name of the form:

*lib***(***member* . . .**)**

refers to a member, or a space-separated list of members, in an **ar**(1) library.

The dependency of the library member on the corresponding file must be given as an explicit entry in the makefile. This can be handled by a pattern matching rule of the form:

*lib***(%.***s***): %.***s*

where *.s* is the suffix of the member; this suffix is typically **.o** for object libraries.

A target name of the form

*lib***((***symbol***))**

refers to the member of a randomized object library that defines the entry point named *symbol*.

Command Execution

Command lines are executed one at a time, *each by its own process or shell.* Shell commands, notably **cd**, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) just before being executed. This is suppressed if it starts with a '**@**', if there is a '**.SILENT:**' entry in the makefile, or if **make** is run with the **–s** option. Although the **–n** option specifies printing without execution, lines containing the macro **$(MAKE)** are executed regardless, and lines containing the **@** special character are printed. The **–t** (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

make invokes the shell with the **–e** (exit-on-errors) argument. Thus, with semicolon-separated command sequences, execution of the later commands depends on the success of the former. This behavior can be overriden by starting the command line with a ' **-**', or by writing a shell script that returns a non-zero status only as it finds appropriate.

Bourne Shell Constructs

To use the Bourne shell **if** control structure for branching, use a command line of the form:

if *expression* ; \
then *command* ; \

```
            . . . ; \
    else command ; \
            . . . ; \
    fi
```

Although composed of several input lines, the escaped NEWLINE characters insure that **make** treats them all as one (shell) command line.

To use the Bourne shell **for** control structure for loops, use a command line of the form:

```
    for var in list ; \
            do command; \
            . . . ;\
    done
```

To refer to a shell variable, use a double-dollar-sign (**$$**). This prevents expansion of the dollar-sign by **make**.

Command Substitutions

To incorporate the standard output of a shell command in a macro, use a definition of the form:

```
    MACRO :sh =command
```

The command is executed only once, standard error output is discarded, and NEWLINE characters are replaced with SPACEs. If the command has a non-zero exit status, **make** halts with an error.

To capture the output of a shell command in a macro reference, use a reference of the form:

```
    $(MACRO :sh)
```

where *MACRO* is the name of a macro containing a valid Bourne shell command line. In this case, the command is executed whenever the reference is evaluated. As with shell command substitutions, the reference is replaced with the standard output of the command. If the command has a non-zero exit status, **make** halts with an error.

Signals

INT, **SIGTERM**, and **QUIT** signals received from the keyboard halt **make** and remove the target file being processed unless that target is in the dependency list for **.PRECIOUS:**.

EXAMPLES

This makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
    pgm: a.o b.o
            $(LINK.c) –o $@ a.o b.o
    a.o: incl.h a.c
            cc –c a.c
    b.o: incl.h b.c
            cc –c b.c
```

The following makefile uses implicit rules to express the same dependencies:

```
pgm: a.o b.o
        cc a.o b.o –o pgm
a.o b.o: incl.h
```

ENVIRONMENT

KEEP_STATE
This environment variable has the same effect as the **.KEEP_STATE:** special-function target. It enables command dependencies, hidden dependencies and writing of the state file.

USE_SVR4_MAKE
This environment variable causes **make** to invoke the generic System V version of **make** (**/usr/ccs/lib/sv4.make**). See **sysV-make**(1).

FILES

/usr/ccs/make	**make**
makefile	
Makefile	current version(s) of **make** description file
SCCS/s.makefile	
SCCS/s.Makefile	SCCS history files for the above makefile(s)
make.rules	default file for user-defined targets, macros, and implicit rules
</usr/share/lib/make/make.rules>	makefile for standard implicit rules and macros (not read if **make.rules** is)
.make.state	state file in the local directory

SEE ALSO

ar(1), **cd**(1), **passwd**(4)

Solaris Advanced User's Guide
Programming Utilities Guide

DIAGNOSTICS

make returns an exit status of **1** when it halts as a result of an error. Otherwise it returns an exit status of **0**.

Don't know how to make target '*target*'
There is no makefile entry for *target*, and none of **make**'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

*** *target* **removed.**
make was interrupted while building *target*. Rather than leaving a partially-completed version that is newer than its dependencies, **make** removes the file named *target*.

*** *target* **not removed.**
make was interrupted while building *target* and *target* was not present in the directory.

*** *target* **could not be removed,** *reason*

make was interrupted while building *target*, which was not removed for the indicated reason.

Read of include file '*file***' failed**

The makefile indicated in an **include** directive was not found, or was inaccessible.

Loop detected when expanding macro value '*macro***'**

A reference to the macro being defined was found in the definition.

Could not write state file '*file***'**

You used the **.KEEP_STATE:** target, but do not have write permission on the state file.

*** **Error code** *n*

The previous shell command returned a nonzero error code.

*** *signal message*

The previous shell command was aborted due to a signal. If '– **core dumped**' appears after the message, a **core** file was created.

Conditional macro conflict encountered

Displayed only when **–d** is in effect, this message indicates that two or more parallel targets currently being processed depend on a target which is built differently for each by virtue of conditional macros. Since the target cannot simultaneously satisfy both dependency relationships, it is conflicted.

BUGS

Some commands return nonzero status inappropriately; to overcome this difficulty, prefix the offending command line in the rule with a '–'.

Filenames with the characters '**=**', '**:**', or '**@**', do not work.

You cannot build **file.o** from **lib(file.o)**.

Options supplied by **MAKEFLAGS** should be reported for nested **make** commands. Use the **–d** option to find out what options the nested command picks up from **MAKEFLAGS**.

This version of **make** is incompatible in certain respects with previous versions:

- The **–d** option output is much briefer in this version. **–dd** now produces the equivalent voluminous output.
- **make** attempts to derive values for the dynamic macros '**$***', '**$<**', and '**$?**', while processing explicit targets. It uses the same method as for implicit rules; in some cases this can lead either to unexpected values, or to an empty value being assigned. (Actually, this was true for earlier versions as well, even though the documentation stated otherwise.)
- **make** no longer searches for SCCS history "(s.)" files.
- Suffix replacement in macro references are now applied after the macro is expanded.

There is no guarantee that makefiles created for this version of **make** will work with earlier versions.

If there is no **make.rules** file in the current directory, and the file **</usr/share/lib/make/make.rules>** is missing, **make** stops before processing any targets. To force **make** to run anyway, create an empty **make.rules** file in the current directory.

Once a dependency is made, **make** assumes the dependency file is present for the remainder of the run. If a rule subsequently removes that file and future targets depend on its existence, unexpected errors may result.

When hidden dependency checking is in effect, the **$?** macro's value includes the names of hidden dependencies. This can lead to improper filename arguments to commands when **$?** is used in a rule.

Pattern replacement macro references cannot be used in the dependency list of a pattern matching rule.

Unlike previous versions, this version of **make** strips a leading '**./**' from the value of the '**$@**' dynamic macro.

With automatic SCCS retrieval, this version of **make** does not support tilde suffix rules.

The only dynamic macro whose value is strictly determined when used in a dependency list is **$@** (takes the form '**$$@**').

make invokes the shell with the **–e** argument. This cannot be inferred from the syntax of the rule alone.

NAME

man – find and display reference manual pages

SYNOPSIS

man [–] [–**adFlrt**] [–**M** *path*] [–**T** *macro-package*] [–**s** *section*] *name* . . .
man [–**M** *path*] –**k** *keyword* . . .
man [–**M** *path*] –**f** *filename* . . .

AVAILABILITY

SUNWdoc

DESCRIPTION

man displays information from the reference manuals. It displays complete manual pages that you select by *name*, or one-line summaries selected either by *keyword* (–**k**), or by the name of an associated file (–**f**).

A *section*, when given, applies to the *name*s that follow it on the command line (up to the next *section*, if any). **man** looks in the indicated section of the manual (or in all sections, if none is specified) for those *name*s; see **Search Paths** below for an explanation of how **man** conducts its search. If no manual page is located, **man** prints an error message.

The reference page sources are typically located in the **/usr/share/man/man**∗ or **/usr/man/man**∗ directories, with each directory corresponding to a section of the manual. Since these directories are optionally installed, they may not reside on your host; you may have to mount **/usr/share/man** from a host on which they do reside. If there are preformatted, up-to-date versions in the corresponding **cat**∗ or **fmt**∗ directories, **man** simply displays or prints those versions. If the preformatted version of interest is out of date or missing, **man** reformats it prior to display and will store the preformatted version if **cat?** or **fmt?** is writable. The **windex** database is not updated. See **catman**(1M). If directories for the preformatted versions are not provided, **man** reformats a page whenever it is requested; it uses a temporary file to store the formatted text during display.

If the standard output is not a terminal, or if the '–' flag is given, **man** pipes its output through **cat**(1); otherwise, **man** pipes its output through **more**(1) to handle paging and underlining on the screen.

OPTIONS

–a	Show all manual pages matching *name* within the **MANPATH** search path. Manual pages are displayed in the order found.
–d	Debug. Displays what a section-specifier evaluates to, method used for searching, and paths searched by **man**.
–F	Force **man** to search all directories specified by **MANPATH** or the **man.cf** file, rather than using the **windex** lookup database. This is useful if the database is not up to date. If the **windex** database does not exist, this option is assumed.
–l	List all manual pages found matching *name* within the search path.
–r	Reformat the manual page, but do not display it. This replaces the **man** – –**t** *name* combination.
–t	**man** arranges for the specified manual pages to be **troff**ed to a suitable raster output device (see **troff**(1). If both the – and –**t** flags are given, **man** updates the **troff**ed versions of each named *name* (if necessary), but does not display them.

–M *path*
Specify an alternate search path for manual pages. *path* is a colon-separated list of directories that contain manual page directory subtrees. For example, if *path* is **/usr/share/man:/usr/local/man**, **man** searches for *name* in the standard location, and then **/usr/local/man**. When used with the **–k** or **–f** options, the **–M** option must appear first. Each directory in the *path* is assumed to contain subdirectories of the form **man**∗, one for each section. This option overrides the **MANPATH** environment variable.

–s *section* ...
Specify sections of the manual for **man** to search. The directories searched for *name* is limited to those specified by *section*. *section* can be a digit (perhaps followed by one or more letters), a word (for example: local, new, old, public), or a letter. To specify multiple sections, separate each section with a comma. This option overrides the **MANPATH** environment variable and the **man.cf** file. See **Search Path** under USAGE for a complete explanation of the default search path order.

–T *macro-package*
Format manual pages using *macro-package* rather than the standard **–man** macros defined in **/usr/share/lib/tmac/an**.

–k *keyword* ...
Print out one-line summaries from the **windex** database (table of contents) that contain any of the given *keyword*s. The **windex** database is created using **catman**(1M).

–f *filename* ...
man attempts to locate manual pages related to any of the given *filename*s. It strips the leading pathname components from each *filename*, and then prints one-line summaries containing the resulting basename or names. This option also uses the **windex** database.

USAGE

Sections

Entries in the reference manuals are organized into *sections*. A section name consists of a major section name, typically a single digit, optionally followed by a subsection name, typically one or more letters. An unadorned major section name acts as an abbreviation for the section of the same name along with all of its subsections. Each section contains descriptions apropos to a particular reference category, with subsections refining these distinctions. See the **intro** manual pages for an explanation of the classification used in this release.

Search Path

Before searching for a given *name*, **man** constructs a list of candidate directories and sections. **man** searches for *name* in the directories specified by the **MANPATH** environment variable. If this variable is not set, **/usr/share/man** is searched by default.

Within the manual page directories, **man** confines its search to the sections specified in the following order:

- *section*s specified on the command line with the **–s** option

- *section*s embedded in the **MANPATH** environment variable
- *section*s specified in the **man.cf** file for each directory specified in the **MANPATH** environment variable

If none of the above exist, **man** searches each directory in the manual page path, and displays the first matching manual page found.

The **man.cf** file has the following format:

MANSECTS=*section*[,*section*] . . .

Lines beginning with ‘#’ and blank lines are considered comments, and are ignored. Each directory specified in **MANPATH** can contain a manual page configuration file, specifying the default search order for that directory.

Formatting Manual Pages

Manual pages are **troff**(1) or **nroff**(1) source files prepared with the –**man** macro package. Please refer to **man**(5) for more information.

Preprocessing Manual Pages

When formatting a manual page, **man** examines the first line to determine whether it requires special processing. If the first line is a string of the form:

´\" *X*

where *X* is separated from the ‘"’ by a single SPACE and consists of any combination of characters in the following list, **man** pipes its input to **troff**(1) or **nroff**(1) through the corresponding preprocessors.

e	**eqn**(1), or **neqn** for **nroff**
r	**refer**(1)
t	**tbl**(1)
v	**vgrind**(1)

If **eqn** or **neqn** is invoked, it will automatically read the file **/usr/pub/eqnchar** (see **eqnchar**(5)). If **nroff**(1) is invoked, **col**(1) is automatically used.

Referring to Other Manual Pages

If the first line of the manual page is a reference to another manual page entry fitting the pattern:

.so man*∗/ *sourcefile*

man processes the indicated file in place of the current one. The reference must be expressed as a pathname relative to the root of the manual page directory subtree.

When the second or any subsequent line starts with **.so**, **man** ignores it; **troff**(1) or **nroff**(1) processes the request in the usual manner.

ENVIRONMENT

MANPATH	A colon-separated list of directories; each directory can be followed by a comma-separated list of sections. If set, its value overrides **/usr/share/man** as the default directory search path, and the **man.cf** file as the default section search path. The –**M** and –**s** flags, in turn, override these values.)
PAGER	A program to use for interactively delivering **man**’s output to the screen. If not set, ‘**more –s**’ (see **more**(1)) is used.

TCAT	The name of the program to use to display **troff**ed manual pages.
TROFF	The name of the formatter to use when the –**t** flag is given. If not set, **troff**(1) is used.

FILES

/usr/share/man	root of the standard manual page directory subtree
/usr/share/man/man?/*	unformatted manual entries
/usr/share/man/cat?/*	**nroff**ed manual entries
/usr/share/man/fmt?/*	**troff**ed manual entries
/usr/share/man/windex	table of contents and keyword database
/usr/share/lib/tmac/an	standard –**man** macro package
/usr/share/lib/pub/eqnchar	standard definitions for **eqn** and **neqn**
man.cf	default search order by section

SEE ALSO

apropos(1), **cat**(1), **col**(1), **eqn**(1), **more**(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1), **vgrind**(1), **whatis**(1), **catman**(1M), **eqnchar**(5), **man**(5)

NOTES

Because **troff** is not 8-bit clean, **man** has not been made 8-bit clean.

The –**f** and –**k** options use the **/usr/share/man/windex** database, which is created by **catman**(1M).

BUGS

The manual is supposed to be reproducible either on a phototypesetter or on an ASCII terminal. However, on a terminal some information (indicated by font changes, for instance) is lost.

Some dumb terminals cannot process the vertical motions produced by the **e** (see **eqn**(1)) preprocessing flag. To prevent garbled output on these terminals, when you use **e** also use **t**, to invoke **col**(1) implicitly. This workaround has the disadvantage of eliminating superscripts and subscripts — even on those terminals that can display them. CTRL-Q will clear a terminal that gets confused by **eqn**(1) output.

NAME

mconnect – connect to SMTP mail server socket

SYNOPSIS

mconnect [**–p** *port*] [**–r**] [*hostname*]

AVAILABILITY

SUNWcsu

DESCRIPTION

mconnect opens a connection to the mail server on a given host, so that it can be tested independently of all other mail software. If no host is given, the connection is made to the local host. Servers expect to speak the Simple Mail Transfer Protocol (SMTP) on this connection. Exit by typing the **quit** command. Typing EOF sends an end of file to the server. An interrupt closes the connection immediately and exits.

OPTIONS

–p *port*	Specify the port number instead of the default SMTP port (number 25) as the next argument.
–r	‘‘Raw’’ mode: disable the default line buffering and input handling. This produces an effect similar to **telnet** to port number 25.

FILES

/etc/mail/sendmail.hf
help file for SMTP commands

SEE ALSO

sendmail(1M)

Postel, Jonathan B *Simple Mail Transfer Protocol*, RFC821 August 1982, SRI Network Information Center

NAME

mcs – manipulate the comment section of an object file

SYNOPSIS

mcs [**–a** *string*] [**–c**] [**–d**] [**–n** *name*] [**–p**] [**–V**] *filename*. . .

DESCRIPTION

The **mcs** command is used to manipulate a section, by default the **.comment** section, in an ELF object file. It is used to add to, delete, print, and compress the contents of a section in an ELF object file, and only print the contents of a section in a COFF object file. **mcs** must be given one or more of the options described below. It applies each of the options in order to each file.

OPTIONS

–a *string*	Append *string* to the comment section of the ELF object files. If *string* contains embedded blanks, it must be enclosed in quotation marks.
–c	Compress the contents of the comment section of the ELF object files. All duplicate entries are removed. The ordering of the remaining entries is not disturbed.
–d	Delete the contents of the comment section from the ELF object files. The section header for the comment section is also removed.
–n *name*	Specify the name of the comment section to access if other than **.comment**. By default, **mcs** deals with the section named **.comment**. This option can be used to specify another section.
–p	Print the contents of the comment section on the standard output. Each section printed is tagged by the name of the file from which it was extracted, using the format *filename***[***member_name***]:** for archive files; and *filename***:** for other files.
–V	Print, on standard error, the version number of **mcs**.

If the input file is an archive (see **ar**(4)), the archive is treated as a set of individual files. For example, if the **–a** option is specified, the string is appended to the comment section of each ELF object file in the archive; if the archive member is not an ELF object file, then it is left unchanged.

If **mcs** is executed on an archive file the archive symbol table will be removed, unless only the **–p** option has been specified. The archive symbol table must be restored by executing the **ar** command with the **–s** option before the archive can be linked by the **ld** command. **mcs** will produce appropriate warning messages when this situation arises.

EXAMPLES

The following example:

```
example% mcs –p filename
```

prints filename’s comment section.

The next example:

```
example% mcs –a string filename
```

appends string to filename’s comment section.

FILES

/tmp/mcs∗	temporary files

SEE ALSO

ar(1), **as**(1), **ld**(1), **tmpnam**(3S), **a.out**(4), **ar**(4)

NOTES

mcs cannot add to, delete or compress the contents of a section that is contained within a segment.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [–**n**] [–**y**]

AVAILABILITY

SUNWcsu

DESCRIPTION

mesg with argument –**n** forbids messages via **write**(1) by revoking non-user write permission on the user's terminal. **mesg** with argument –**y** reinstates permission. All by itself, **mesg** reports the current state without changing it.

OPTIONS

–**n**	Forbids messages via **write**(1).
–**y**	Reinstates permission.

FILES

/dev/tty*

SEE ALSO

write(1)

DIAGNOSTICS

Exit status is **0** if messages are receivable, **1** if not, **2** on error.

NAME

message – puts its arguments on FMLI message line

SYNOPSIS

message [–**t**] [–**b** [*num*]] [–**o**] [–**w**] [*string*]
message [–**f**] [–**b** [*num*]] [–**o**] [–**w**] [*string*]
message [–**p**] [–**b** [*num*]] [–**o**] [–**w**] [*string*]

DESCRIPTION

The **message** command puts *string* out on the FMLI message line. If there is no string, the *stdin* input to **message** will be used. The output of **message** has a duration (length of time it remains on the message line). The default duration is "transient": it or one of two other durations can be requested with the mutually-exclusive options below.

Messages displayed with **message** –**p** will replace (change the value of) any message currently displayed or stored via use of the **permanentmsg** descriptor. Likewise, **message** –**f** will replace any message currently displayed or stored via use of the **framemsg** descriptor. If more than one message in a frame definition file is specified with the **-p** option, the last one specified will be the permanent duration message.

The *string* argument should always be the last argument.

OPTIONS

–**t**	Explicitly defines a message to have transient duration. Transient messages remain on the message line only until the user presses another key or a **CHECKWORLD** occurs. The descriptors **itemmsg , fieldmsg , invalidmsg , choicemsg ,** the default-if-not-defined value of **oninterrupt ,** and FMLI generated error messages (that is, from syntax errors) also output transient duration messages. Transient messages take precedence over both frame messages and permanent messages.
–**f**	Defines a message to have "frame" duration. Frame messages remain on the message line as long as the frame in which they are defined is current. The descriptor **framemsg** also outputs a frame duration message. Frame messages take precedence over permanent messages.
–**p**	Defines a message to have "permanent" duration. Permanent messages remain on the message line for the length of the FMLI session, unless explicitly replaced by another permanent message or temporarily superseded by a transient message or frame message. A permanent message is not affected by navigating away from, or by closing, the frame which generated the permanent message. The descriptor **permanentmsg** also outputs a permanent duration message.
–**b***[num]*	Rings the terminal bell *num* times, where *num* is an integer from 1 to 10. The default value is 1. If the terminal has no bell, the screen will flash *num* times instead, if possible.
–**o**	Forces **message** to duplicate its message to *stdout* .
-w	Turns on the working indicator.

EXAMPLES

When a value entered in a field is invalid, ring the bell 3 times and then display **Invalid Entry: Try again!** on the message line:

```
invalidmsg=`message –b 3 "Invalid Entry: Try again!"`
```

Display a message that tells the user what is being done:

```
done=`message EDITOR has been set in your environment` close
```

Display a message on the message line and *stdout* for each field in a form (a pseudo-"field duration" message).

```
fieldmsg="`message -o -f "Enter a filename."`"
```

Display a blank transient message (effect is to "remove" a permanent or frame duration message).

```
done=`message  ""` nop
```

SEE ALSO

sleep(1)

NOTES

If **message** is coded more than once on a single line, it may appear that only the right-most instance is interpreted and displayed. Use **sleep**(1) between uses of **message** in this case, to display multiple messages.

message -f should not be used in a stand-alone backquoted expression or with the **init** descriptor because the frame is not yet current when these are evaluated.

In cases where **`message -f** "*string*"` is part of a stand-alone backquoted expression, the context for evaluation of the expression is the previously current frame. The previously current frame can be the frame that issued the **open** command for the frame containing the backquoted expression, or it can be a frame given as an argument when **fmli** was invoked. That is, the previously current frame is the one whose frame message will be modified.

Permanent duration messages are displayed when the user navigates to the command line.

NAME

mkdir – make directories

SYNOPSIS

mkdir [**–m** *mode*] [**–p**] *dirname*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

mkdir creates the named directories in mode 777 (possibly altered by **umask**(1)).

Standard entries in a directory (for instance, the files **.**, for the directory itself, and **..**, for its parent) are made automatically. **mkdir** cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner-ID and group-ID of the new directories are set to the process's effective user-ID and group-ID, respectively. The behavior mimics the **mkdir**(2) system call.

OPTIONS

–m *mode* This option allows users to specify the mode to be used for new directories. Choices for modes can be found in **chmod**(1).

–p With this option, **mkdir** creates **dirname** by creating all the non-existing parent directories first.

EXAMPLES

The following example:

```
example% mkdir -p ltr/jd/jan
```

creates the subdirectory structure **ltr/jd/jan**.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **mkdir** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **mkdir** behaves.

LC_CTYPE

Determines how **mkdir** handles characters. When **LC_CTYPE** is set to a valid value, **mkdir** can display and handle text and filenames containing valid characters for that locale. **mkdir** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **mkdir** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

rm(1), **sh**(1), **umask**(1), **intro**(2), **mkdir**(2), **environ**(5)

DIAGNOSTICS

mkdir returns exit code **0** if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero.

NAME

mkmsgs – create message files for use by gettxt

SYNOPSIS

mkmsgs [–**o**] [–**i** *locale*] *inputstrings msgfile*

AVAILABILITY

SUNWloc

DESCRIPTION

The **mkmsgs** utility is used to create a file of text strings that can be accessed using the text retrieval tools (see **gettxt**(1), **srchtxt**(1), **exstr**(1), and **gettxt**(3C)). It will take as input a file of text strings for a particular geographic locale (see **setlocale**(3C)) and create a file of text strings in a format that can be retrieved by both **gettxt**(1) and **gettxt**(3C). By using the –**i** option, you can install the created file under the **/usr/lib/locale/***locale***/LC_MESSAGES** directory (*locale* corresponds to the language in which the text strings are written).

inputstrings is the name of the file that contains the original text strings. *msgfile* is the name of the output file where **mkmsgs** writes the strings in a format that is readable by **gettxt**(1) and **gettxt**(3C). The name of *msgfile* can be up to 14 characters in length, but may not contain either \ **0** (null) or the ASCII code for **/** (slash) or **:** (colon).

The input file contains a set of text strings for the particular geographic locale. Text strings are separated by a newline character. Nongraphic characters must be represented as alphabetic escape sequences. Messages are transformed and copied sequentially from *inputstrings* to *msgfile*. To generate an empty message in *msgfile*, leave an empty line at the correct place in *inputstrings*.

Strings can be changed simply by editing the file *inputstrings*. New strings must be added only at the end of the file; then a new *msgfile* file must be created and installed in the correct place. If this procedure is not followed, the retrieval function will retrieve the wrong string and software compatibility will be broken.

OPTIONS

–**o**	Overwrite *msgfile*, if it exists.
–**i** *locale*	Install *msgfile* in the **/usr/lib/locale/***locale***/LC_MESSAGES** directory. Only someone who is super-user or a member of group **bin** can create or overwrite files in this directory. Directories under **/usr/lib/locale** will be created if they do not exist.

EXAMPLES

The following example shows an input message source file **C.str**:

```
File %s:\t cannot be opened\n
%s: Bad directory\n
        .
        .
        .
write error\n
        .
        .
```

The following command uses the input strings from **C.str** to create text strings in the appropriate format in the file **UX** in the current directory:

```
example% mkmsgs C.str UX
```

The following command uses the input strings from **FR.str** to create text strings in the appropriate format in the file **UX** in the directory **/usr/lib/locale/french/LC_MESSAGES/UX**:

```
example% mkmsgs –i french FR.str UX
```

These text strings would be accessed if you had set the environment variable **LC_MESSAGES=french** and then invoked one of the text retrieval tools listed at the beginning of the **DESCRIPTION** section.

FILES

/usr/lib/locale/*locale***/LC_MESSAGES/***	message files created by **mkmsgs**

SEE ALSO

exstr(1), **gettxt**(1), **srchtxt**(1), **gettxt**(3C), **setlocale**(3C)

NAME

mkstr – create an error message file by massaging C source files

SYNOPSIS

/usr/ucb/mkstr [–] *messagefile prefix filename*. . .

AVAILABILITY

SUNWscpu

DESCRIPTION

mkstr creates files of error messages. You can use **mkstr** to make programs with large numbers of error diagnostics much smaller, and to reduce system overhead in running the program — as the error messages do not have to be constantly swapped in and out.

mkstr processes each of the specified *filename*s, placing a massaged version of the input file in a file with a name consisting of the specified *prefix* and the original source file name. A typical example of using **mkstr** would be:

```
mkstr pistrings processed *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file **pistrings** and processed copies of the source for these files to be placed in files whose names are prefixed with *processed*.

To process the error messages in the source to the message file, **mkstr** keys on the string ‘**error("**’ in the input stream. Each time it occurs, the C string starting at the ‘"’ is placed in the message file followed by a null character and a NEWLINE character; the null character terminates the message so it can be easily used when retrieved, the NEWLINE character makes it possible to sensibly **cat** the error message file to see its contents. The massaged copy of the input file then contains a **lseek** pointer into the file which can be used to retrieve the message, that is:

```
        char efilname[ ] = "/usr/lib/pi_strings";
        int efil = –1;

        error(a1, a2, a3, a4)
        {
                char
                buf[256];
                if (efil < 0) {
                                efil = open(efilname, 0);
                                if (efil < 0) {
oops:
                                perror (efilname);
                                exit (1);
```

```
				}
			}
			if (lseek(efil, (long) a1, 0) | | read(efil, buf, 256) <= 0)
				goto oops;
			printf(buf, a2, a3, a4);
		}
```

OPTIONS

– Place error messages at the end of the specified message file for recompiling part of a large **mkstr**ed program.

SEE ALSO

xstr(1)

NAME

more, page – browse or page through a text file

SYNOPSIS

more [**–cdflrsuw**] [*–lines*] [**+***linenumber*] [**+/***pattern*] [*filename* . . .]

page [**–cdflrsuw**] [*–lines*] [**+***linenumber*] [**+/***pattern*] [*filename* . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

more is a filter that displays the contents of a text file on the terminal, one screenful at a time. It normally pauses after each screenful, and prints **--More--** at the bottom of the screen. **more** provides a two-line overlap between screens for continuity. If **more** is reading from a file rather than a pipe, the percentage of characters displayed so far is also shown.

more scrolls up to display one more line in response to a RETURN character; it displays another screenful in response to a SPACE character. Other commands are listed below.

page clears the screen before displaying the next screenful of text; it only provides a one-line overlap between screens.

more sets the terminal to *noecho* mode, so that the output can be continuous. Commands that you type do not normally show up on your terminal, except for the **/** and **!** commands.

If the standard output is not a terminal, **more** acts just like **cat**(1), except that a header is printed before each file in a series.

OPTIONS

The following options are available with **more**:

–c Clear before displaying. Redrawing the screen instead of scrolling for faster displays. This option is ignored if the terminal does not have the ability to clear to the end of a line.

–d Display error messages rather than ringing the terminal bell if an unrecognized command is used. This is helpful for inexperienced users.

–f Do not fold long lines. This is useful when lines contain nonprinting characters or escape sequences, such as those generated when **nroff**(1) output is piped through **ul**(1).

–l Do not treat FORMFEED characters (CTRL-L) as “page breaks.” If **–l** is not used, **more** pauses to accept commands after any line containing a **ˆL** character (CTRL-L). Also, if a file begins with a FORMFEED, the screen is cleared before the file is printed.

–r Normally, **more** ignores control characters that it does not interpret in some way. The **–r** option causes these to be displayed as *ˆC* where *C* stands for any such control character.

–s Squeeze. Replace multiple blank lines with a single blank line. This is helpful when viewing **nroff**(1) output, on the screen.

–u	Suppress generation of underlining escape sequences. Normally, **more** handles underlining, such as that produced by **nroff**(1), in a manner appropriate to the terminal. If the terminal can perform underlining or has a stand-out mode, **more** supplies appropriate escape sequences as called for in the text file.
–w	Normally, **more** exits when it comes to the end of its input. With –**w**, however, **more** prompts and waits for any key to be struck before exiting.
–*lines*	Display the indicated number of *lines* in each screenful, rather than the default (the number of lines in the terminal screen less two).
+*linenumber*	Start up at *linenumber*.
+/*pattern*	Start up two lines above the line containing the regular expression *pattern*. Note: Unlike editors, this construct should *not* end with a '/'. If it does, then the trailing slash is taken as a character in the search pattern.

USAGE

Environment

more uses the terminal's **/usr/share/lib/terminfo** entry to determine its display characteristics, and looks in the environment variable **MORE** for any preset options. For instance, to page through files using the –**c** mode by default, set the value of this variable to –**c**. (Normally, the command sequence to set up this environment variable is placed in the **.login** or **.profile** file).

Commands

The commands take effect immediately; it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may type the line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the '**--More--(***xx***%)**' message.

In the following commands, *i* is a numerical argument (**1** by default).

*i*SPACE	Display another screenful, or *i* more lines if *i* is specified.
*i*RETURN	Display another line, or *i* more lines, if specified.
*i***^D**	(CTRL–D) Display (scroll down) 11 more lines. If *i* is given, the scroll size is set to *i*.
*i*d	Same as **^D**.
*i***z**	Same as SPACE, except that *i*, if present, becomes the new default number of lines per screenful.
*i***s**	Skip *i* lines and then print a screenful.
*i***f**	Skip *i* screenfuls and then print a screenful.
*i***^B**	(CTRL-B) Skip back *i* screenfuls and then print a screenful.
b	Same as **^B** (CTRL-B).
q **Q**	Exit from **more**.
=	Display the current line number.

v	Drop into the editor indicated by the **EDITOR** environment variable, at the current line of the current file. The default editor is **ed**(1).
h	Help. Give a description of all the **more** commands.
i/*pattern*	Search forward for the *i* th occurrence of the regular expression *pattern*. Display the screenful starting two lines before the line that contains the *i* th match for the regular expression *pattern*, or the end of a pipe, whichever comes first. If **more** is displaying a file and there is no such match, its position in the file remains unchanged. Regular expressions can be edited using erase and kill characters. Erasing back past the first column cancels the search command.
*i***n**	Search for the *i* th occurrence of the last *pattern* entered.
´	Single quote. Go to the point from which the last search started. If no search has been performed in the current file, go to the beginning of the file.
!*command*	Invoke a shell to execute *command* . The characters **%** and **!**, when used within *command* are replaced with the current filename and the previous shell command, respectively. If there is no current filename, **%** is not expanded. Prepend a backslash to these characters to escape expansion.
i:**n**	Skip to the *i* th next filename given in the command line, or to the last filename in the list if *i* is out of range.
i:**p**	Skip to the *i* th previous filename given in the command line, or to the first filename if *i* is out of range. If given while **more** is positioned within a file, go to the beginning of the file. If **more** is reading from a pipe, **more** simply rings the terminal bell.
:f	Display the current filename and line number.
:q **:Q**	Exit from **more** (same as **q** or **Q**).
.	Dot. Repeat the previous command.
^	Halt a partial display of text. **more** stops sending output, and displays the usual **--More--** prompt. Unfortunately, some output is lost as a result.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **more** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **more** behaves.

LC_CTYPE

Determines how **more** handles characters. When **LC_CTYPE** is set to a valid value, **more** can display and handle text and filenames containing valid characters for that locale. **more** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **more** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/usr/lib/more.help	help file
/usr/share/lib/terminfo/*	terminal data base

SEE ALSO

cat(1), **csh**(1), **man**(1), **script**(1), **sh**(1), **environ**(4), **environ**(5)

NOTES

Skipping backwards is too slow on large files.

NAME

msgfmt – create a message object from a message file

SYNOPSIS

msgfmt [**–v**] [**–o** *output-file*] *filename***.po** . . .

AVAILABILITY

SUNWloc

DESCRIPTION

msgfmt creates message object files from portable object files (*filename***.po**), without changing the portable object files.

The **.po** file contains messages displayed to users by system commands or by application programs. **.po** files can be edited, and the messages in them can be rewritten in any language supported by the system.

The **xgettext**(1) command can be used to create **.po** files from script or programs.

Portable Object Files

Formats for all **.po** files are the same. Each **.po** file contains one or more lines, with each line containing either a comment or a statement. Comments start the line with a hash mark (#) and end with the newline character. All comments are ignored. The format of a statement is:

directive value

Each directive starts at the beginning of the line and is separated from *value* by white space (such as one or more space or tab characters). *value* consists of one or more quoted strings separated by white space. Use any of the following types of directives:

domain *domainname*
msgid *message_identifier*
msgstr *message_string*

The behavior of the **domain** directive is affected by the options used. See OPTIONS for the behavior when the **–o** option is specified. If the **–o** option is not specified, the behavior of the **domain** directive is as follows:

- All *msgids* from the beginning of each **.po** file to the first domain directive are put into a default message object file, **messages.mo**.
- When **msgfmt** encounters a **domain** *domainname* directive in the **.po** file, all following *msgids* until the next **domain** directive are put into the message object file *domainname***.mo**.
- Duplicate *msgids* are defined in the scope of each domain. That is, a *msgid* is considered a duplicate only if the identical *msgid* exists in the same domain.
- All duplicate *msgids* are ignored.

The **msgid** directive specifies the value of a message identifier associated with the directive that follows it. The *message_identifier* string identifies a target string to be used at retrieval time. Each statement containing a **msgid** directive must be followed by a statement containing a **msgstr** directive.

The **msgstr** directive specifies the target string associated with the *message_identifier* string declared in the immediately preceding **msgid** directive.

Message strings can contain the escape sequences **\n** for newline, **\t** for tab, **\v** for vertical tab, **\b** for backspace, **\r** for carriage return, **\f** for formfeed, \\ for backslash, \" for double quote, **\ddd** for octal bit pattern, and **\xDD** for hexadecimal bit pattern.

OPTIONS

–v Verbose. List duplicate message identifiers. Message strings are not redefined.

–o *output-file* Specify output file name as *output-file*. All **domain** directives and duplicate *msgids* in the **.po** file are ignored.

EXAMPLES

In this example **module1.po** and **module2.po** are portable message objects files.

```
example% cat module1.po
# default domain "messages.mo"
msgid  "msg 1"
msgstr "msg 1 translation"
#
domain "help_domain"
msgid  "help 2"
msgstr "help 2 translation"
#
domain "error_domain"
msgid  "error 3"
msgstr "error 3 translation"

example% cat module2.po
# default domain "messages.mo"
msgid  "mesg 4"
msgstr "mesg 4 translation"
#
domain "error_domain"
msgid  "error 5"
msgstr "error 5 translation"
#
domain "window_domain"
msgid  "window 6"
msgstr "window 6 translation"
```

The following command will produce the output files, **messages.mo**, **help_domain.mo**, and **error_domain.mo**.

```
example% msgfmt module1.po
```

The following command will produce the output files, **messages.mo**, **help_domain.mo**, **error_domain.mo**, and **window_domain.mo**.

```
example% msgfmt module1.po module2.po
```

The following example will produce the output file **hello.mo**.

```
example% msgfmt –o hello.mo module1.po module2.po
```

Install message object files in **/usr/lib/locale/***locale***/LC_MESSAGES/***domain***.mo** where *locale* is the message locale as set by **setlocale**(3C), and *domain* is text domain as set by **textdomain( )**. The **/usr/lib/locale** portion can optionally be changed by calling **bindtextdomain( )**. See **gettext**(3I).

SEE ALSO

xgettext(1), **gettext**(3I)

NOTES

Neither **msgfmt** nor any **gettext**(3I) routine imposes a limit on the total length of a message. However, each line in the ∗**.po** file is limited to **MAX_INPUT** (512) bytes.

Installing message catalogs under the C locale is pointless, since they are ignored for the sake of efficiency.

NAME

mt – magnetic tape control

SYNOPSIS

mt [**–f** *tapename*] *command*... [*count*]

AVAILABILITY

SUNWcsu

DESCRIPTION

mt sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable **TAPE** is used. If **TAPE** does not exist, **mt** uses the device **/dev/rmt/0**. *tapename* refers to a raw tape device. By default, **mt** performs the requested operation once; multiple operations may be performed by specifying *count*.

The available *command*s are listed below. Only as many characters as are required to uniquely identify a *command* need be specified.

mt returns a **0** exit status when the operation(s) were successful, **1** if the command was unrecognized or if **mt** was unable to open the specified tape drive, and **2** if an operation failed.

mt Commands

eof, **weof**	Write *count* EOF marks at the current position on the tape.
fsf	Forward space over *count* EOF marks. The tape is positioned on the first block of the file.
fsr	Forward space *count* records.
bsf	Back space over *count* EOF marks. The tape is positioned on the beginning-of-tape side of the EOF mark.
bsr	Back space *count* records.
nbsf	Back space *count* files. The tape is positioned on the first block of the file. This is equivalent to *count+1* **bsf**’s followed by one **fsf**.
asf	Absolute space to *count* file number. This is equivalent to a **rewind** followed by a **fsf** *count*.

For the following commands, *count* is ignored:

eom	Space to the end of recorded media on the tape. This is useful for appending files onto previously written tapes.
rewind	Rewind the tape.
offline, **rewoffl**	Rewind the tape and, if appropriate, take the drive unit off-line by unloading the tape.
status	Print status information about the tape unit.
retension	Rewind the cartridge tape completely, then wind it forward to the end of the reel and back to beginning-of-tape to smooth out tape tension.
erase	Erase the entire tape.

FILES

/dev/rmt/∗	magnetic tape interface
/dev/rmt/∗b	
/dev/rmt/∗bn	
/dev/rmt/∗c	
/dev/rmt/∗cb	
/dev/rmt/∗cbn	
/dev/rmt/∗cn	
/dev/rmt/∗h	
/dev/rmt/∗hb	
/dev/rmt/∗hbn	
/dev/rmt/∗hn	
/dev/rmt/∗l	
/dev/rmt/∗lb	
/dev/rmt/∗lbn	
/dev/rmt/∗ln	
/dev/rmt/∗m	
/dev/rmt/∗mb	
/dev/rmt/∗mbn	
/dev/rmt/∗mn	
/dev/rmt/∗n	
/dev/rmt/∗u	
/dev/rmt/∗ub	
/dev/rmt/∗ubn	
/dev/rmt/∗un	

SEE ALSO

tar(1), **tcopy**(1), **ar**(4), **environ**(4), **mtio**(7), **st**(7)

BUGS

Not all devices support all options. Some options are hardware-dependent. Refer to the corresponding device manual page.

mt is architecture sensitive. Heterogeneous operation (that is, Sun3 to Sun4 or visa versa) is not supported.

NAME

mv – move files

SYNOPSIS

mv [**–f**] [**–i**] *filename1* [*filename2* . . .] *target*

AVAILABILITY

SUNWcsu

DESCRIPTION

The **mv** command moves *filenamen* to *target*. *filenamen* and *target* may not have the same name. (Care must be taken when using **sh**(1) metacharacters). If *target* is not a directory, only one file may be specified before it; if it is a directory, more than one file may be specified. If *target* does not exist, **mv** creates a file named *target*. If *target* exists and is not a directory, its contents are overwritten. If *target* is a directory the file(s) are moved to that directory.

If **mv** determines that the mode of *target* forbids writing, it will print the mode (see **chmod**(2)), ask for a response, and read the standard input for one line. If the line begins with **y**, the **mv** occurs, if permissible; otherwise, the command exits. When the parent directory of *filenamen* is writable and has the sticky bit set, one or more of the following conditions must be true:

the user must own the file
the user must own the directory
the file must be writable by the user
the user must be a privileged user

If *filenamen* is a directory, *target* must be a directory in the same physical file system. *target* and *filenamen* do not have to share the same parent directory.

If *filenamen* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

OPTIONS

–f	**mv** will move the file(s) without prompting even if it is writing over an existing *target*. This option overrides the **–i** option. Note that this is the default if the standard input is not a terminal.
–i	**mv** will prompt for confirmation whenever the move would overwrite an existing *target* . A **y** answer means that the move should proceed. Any other answer prevents **mv** from overwriting the *target*.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **mv** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **mv** behaves.

LC_CTYPE

Determines how **mv** handles characters. When **LC_CTYPE** is set to a valid value, **mv** can display and handle text and filenames containing valid characters for that locale. **mv** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **mv** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

chmod(1), **cp**(1), **cpio**(1), **ln**(1), **rm**(1), **environ**(5)

NOTES

If *filenamen* and *target* are on different file systems, **mv** copies the file and deletes the original; any links to other files are lost.

A –– permits the user to mark explicitly the end of any command line options, allowing **mv** to recognize filename arguments that begin with a –. As an aid to BSD migration, **mv** will accept – as a synonym for ––. This migration aid may disappear in a future release. If a –– and a – both appear on the same command line, the second will be interpreted as a filename.

NAME

nawk – pattern scanning and processing language

SYNOPSIS

nawk [–**F** *re*] [–**v** *var=value*] [**'***prog***'**] [*filename* ...]
nawk [–**F** *re*] [–**v** *var=value*] [–**f** *program-file*] [*filename* ...]

AVAILABILITY

SUNWesu

DESCRIPTION

nawk scans each input *filename* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (′) to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *filename* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the –**f** *program-file* option. Input files are read in order; if there are no files, the standard input is read. The file name **'–'** means the standard input.

OPTIONS

–**F** *re*	Specifies the input field separator. *re* can be either a character or a regular expression.
–**v** *var=value*	Allows the setting of a variable to *value*, before executing any part of the **nawk** script. The –**v** must be specified for each *variable* to be set.
–**f** *program-file*	Specifiy a file, *program-file*, to be used as the source for the program.

USAGE

Input Lines

Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. Any *filename* of the form *var=value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. *Variables* assigned in this manner are not available inside a **BEGIN** rule, and are assigned after previously specified files have been read. The option –**v** followed by *var=value* is an assignment to be done before *prog* is executed; any number of –**v** options may be present.

An input line is normally made up of fields separated by white spaces. (This default can be changed by using the **FS** built-in variable or the –**F** *re* option.) The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if **FS** is assigned a value that does not include any of the white spaces, then leading blanks are not ignored. The fields are denoted **$1**, **$2**, **...**; **$0** refers to the entire line.

Pattern-action Statements

A pattern-action statement has the form:

pattern **{** *action* **}**

Either pattern or action may be omitted. If there is no action, the matching line is printed. If there is no pattern, the action is performed on every input line. Pattern-action statements are separated by newlines or semicolons.

Patterns are arbitrary Boolean combinations (**!**, || , **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

expression relop expression
expression matchop regular_expression
expression **in** *array-name*
(*expression***,***expression***,** ... **) in** *array-name*

where a *relop* is any of the six relational operators in C, and a *matchop* is either **~** (contains) or **!~** (does not contain). An *expression* is an arithmetic expression, a relational expression, the special expression

var **in** *array*

or a Boolean combination of these.

Regular expressions are as in **egrep**(1). In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between the occurrence of the first pattern to the occurrence of the second pattern.

The special patterns **BEGIN** and **iEND** may be used to capture control before the first input line has been read and after the last input line has been read respectively. These keywords do not combine with any other patterns.

Built-in Variables

Built-in variables include:

ARGC	command line argument count
ARGV	command line argument array
ENVIRON	array of environment variables; subscripts are names
FILENAME	name of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression (default blank and tab)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default **%.6g)**
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RLENGTH	length of string matched with the **match() function**
RS	input record separator (default new-line)
RSTART	starting position of string matched with the **match() function**
SUBSEP	separates multiple subscripts (default is 034)

An action is a sequence of statements. A statement may be one of the following:

if (*expression*) *statement* [**else** *statement*]
while (*expression*) *statement*
do *statement* **while** (*expression*)
for (*expression* ; *expression* ; *expression*) *statement*

for (*var* **in** *array*) *statement*
delete *array*[*subscript*] #delete an array element
break
continue
{ [*statement*] . . . }
expression # commonly variable = expression
print [*expression-list*] [>*expression*]
printf *format* [, *expression-list*] [>*expression*]
next # skip remaining patterns on this input line
exit [**expr**] # skip the rest of the input; exit status is expr
return [**expr**]

Statements are terminated by semicolons, newlines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators **+**, **–**, **∗**, **/**, **%**, **ˆ** and concatenation (indicated by a blank). The operators **++ –– += –= ∗= /= %= ˆ= > >= < <= == != ?:** are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as **[i,j,k]** are permitted; the constituents are concatenated, separated by the value of **SUBSEP**. String constants are quoted (""), with the usual C escapes recognized within.

The **print** statement prints its arguments on the standard output, or on a file if **>***expression* is present, or on a pipe if '|*cmd*' is present. The output resulted from the print statement is terminated by the output record separator with each argument separated by the current output field separator. The **printf** statement formats its expression list according to the format (see **printf**(3S)). The built-in function **close**(*expr*) closes the file or pipe *expr*.

The mathematical functions: **atan2**, **cos**, **exp**, **log**, **sin**, **sqrt**, are built-in.

Other built-in functions include:

gsub(*for*,*repl*,*in***)**	behaves like **sub** (see below), except that it replaces successive occurrences of the regular expression (like the **ed** global substitute command).
index(*s*, *t***)**	returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.
int(*s***)**	truncates *s* to an integer value. If *s* is not specified, $0 is used.
length(*s***)**	returns the length of its argument taken as a string, or of the whole line if there is no argument.
match(*s*, *re***)**	returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. **RSTART** is set to the starting position (which is the same as the returned value), and **RLENGTH** is set to the length of the matched string.
rand	random number on (0, 1).
split(*s*, *a*, *fs***)**	splits the string *s* into array elements *a*[*1*], *a*[*2*], . . . *a*[*n*], and returns *n*.

The separation is done with the regular expression *fs* or with the field separator **FS** if *fs* is not given.

srand	sets the seed for **rand**.
sprintf(*fmt***,** *expr***,** *expr***, . . .)**	formats the expressions according to the **printf**(3S) format given by *fmt* and returns the resulting string.
sub(*for***,***repl***,***in***)**	substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, **nawk** substitutes in the current record (**$0**).
substr(*s***,***m***,***n***)**	returns the *n*-character substring of *s* that begins at position *m*.

The input / output built-in functions are:

close(*filename***)**	closes the file or pipe named *filename*.
cmd **| getline**	pipes the output of *cmd* into **getline** each successive call to **getline** returns the next line of output from *cmd*.
getline	sets **$0** to the next input record from the current input file.
getline <*filename*	sets **$0** to the next record from *file*.
getline *x*	sets variable *x* instead.
getline *x* **<***filename*	sets *x* from the next record of *file*.
system(*cmd***)**	executes *cmd* and returns its exit status.

All forms of **getline** return 1 for successful input, 0 for end of file, and –1 for an error.

nawk also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

function *name***(***args***,...) {** *stmts* **}**

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and / or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average:

```
        { s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; −−i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Simulate **echo**(1):

```
BEGIN {
        for (i = 1; i < ARGC; i++)
                printf "%s", ARGV[i]
        printf "\n"
        exit
        }
```

Print a file, filling in page numbers starting at 5:

```
/Page/  { $2 = n++; }
        { print }
```

Assuming this program is in a file named **prog**, the following command line prints the file **input** numbering its pages starting at 5: **nawk −f prog n=5 input**.

SEE ALSO

egrep(1), **grep**(1), **sed**(1), **printf**(3S)

The **awk** chapter in the *Solaris Advanced User's Guide*.

A. V. Aho, B. W. Kerninghan, P. J. Weinberger, *The AWK Programming Language* Addison-Wesley, 1988.

NOTES

nawk is a new version of **awk** that provides capabilities unavailable in previous versions. This version will become the default version of **awk** in the next major release.

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

AVAILABILITY

SUNWnisu

DESCRIPTION

newaliases rebuilds the random access data base for the mail aliases file **/etc/aliases**. It is run automatically by **sendmail**(1M) (in the default configuration) whenever a message is sent.

FILES

/etc/aliases

SEE ALSO

sendmail(1M), **aliases**(4)

NAME

newform – change the format of a text file

SYNOPSIS

newform [–**s**] [–**i***tabspec*] [–**o***tabspec*] [–**b***n*] [–**e***n*] [–**p***n*] [–**a***n*] [–**f**] [–**c***char*] [–**l***n*] [*filename*. . .]

AVAILABILITY

SUNWesu

DESCRIPTION

newform reads lines from the named *filenames*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for –**s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *filenames*. Command line options are processed in the order specified. This means that option sequences like ‘‘–**e**15 –**l**60’’ will yield results different from ‘‘–**l**60 –**e**15’’. Options are applied to all *filenames* on the command line.

OPTIONS

–**s** Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a ∗ and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform –**s** –**i** –**l** –**a** –**e** *file-name*

–**i***tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in **tabs**(1). In addition, *tabspec* may be ––, in which **newform** assumes that the tab specification is to be found in the first line read from the standard input (see **fspec**(4)). If no *tabspec* is given, *tabspec* defaults to –**8**. A *tabspec* of –**0** expects no tabs; if any are found, they are treated as –**1**.

–**o***tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for –**i***tabspec*. If no *tabspec* is given, *tabspec* defaults to –**8**. A *tabspec* of –**0** means that no spaces will be converted to tabs on output.

–**b***n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see –**l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when –**b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

newform –l1 –b7 *file-name*

–e*n*	Same as **–b***n* except that characters are truncated from the end of the line.
–p*n*	Prefix *n* characters (see **–c***char*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.
–a*n*	Same as **–p***n* except characters are appended to the end of a line.
–f	Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **–o** option. If no **–o** option is specified, the line which is printed will contain the default specification of **–8**.
–c*char*	Change the prefix/append character to *char*. Default character for *char* is a space.
–l*n*	Set the effective line length to *n* characters. If *n* is not entered, **–l** defaults to 72. The default line length without the **–l** option is 80 characters. Note: Tabs and backspaces are considered to be one character (use **–i** to expand tabs to spaces). The **–l1** must be used to set the effective line length shorter than any existing line in the file so that the **–b** option is activated.

SEE ALSO

csplit(1), **tabs**(1), **fspec**(4)

DIAGNOSTICS

All diagnostics are fatal.

usage: ...
  newform was called with a bad option.

"**not –s format**"
  There was no tab on one line.

"**can't open file**"
  Self-explanatory.

"**internal line too long**"
  A line exceeds 512 characters after being expanded in the internal work buffer.

"**tabspec in error**"
  A tab specification is incorrectly formatted, or specified tab stops are not ascending.

"**tabspec indirection illegal**"
  A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0 – normal execution
1 – for any error

NOTES

newform normally only keeps track of physical characters; however, for the –**i** and –**o** options, **newform** will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of –**i**–– or –**o**––).

If the –**f** option is used, and the last –**o** option specified was –**o**––, and was preceded by either a –**o**–– or a –**i**––, the tab specification format line will be incorrect.

NAME

newgrp – shell built-in function to allow new group permissions to the user

SYNOPSIS

sh

newgrp [*argument*]

ksh

† **newgrp** [*arg* . . .]

DESCRIPTION

sh

Equivalent to **exec newgrp** *argument*. See **newgrp**(1M) for usage and description.

ksh

Equivalent to **exec /bin/newgrp** *arg* . . ..

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

ksh(1), **sh**(1), **newgrp**(1M)

NAME

news – print news items

SYNOPSIS

news [**–a**] [**–n**] [**–s**] [*items*]

AVAILABILITY

SUNWesu

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/var/news**.

When invoked without arguments, **news** prints the contents of all current files in **/var/news**, most recent first, with each preceded by an appropriate header. **news** stores the ‘‘currency’’ time as the modification date of a file named **.news_time** in the user’s home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered ‘‘current.’’

OPTIONS

–a **news** prints all items, regardless of currency. In this case, the stored time is not changed.

–n **news** reports the names of the current items without printing their contents, and without changing the stored time.

–s **news** reports how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of **news** in one’s **.profile** file, or in the system’s **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **news** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **news** behaves.

LC_CTYPE

Determines how **news** handles characters. When **LC_CTYPE** is set to a valid value, **news** can display and handle text and filenames containing valid characters for that locale. **news** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **news** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

FILES

/etc/profile
/var/news/*
$HOME/.news_time

SEE ALSO

profile(4), **environ**(5)

NAME

nice – run a command at low priority

SYNOPSIS

/usr/bin/nice [*–increment*] *command* [*arguments*]

AVAILABILITY

SUNWcsu

DESCRIPTION

/usr/bin/nice executes *command* with a lower CPU scheduling priority. The **priocntl** command is a more general interface to scheduler functions.

The invoking process (generally the user's shell) must be in a scheduling class that supports the **/usr/bin/nice** command.

OPTIONS

–increment	If the *increment* argument (in the range 1–19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, for example, —**10**. A negative increment assigned by an unprivileged user is ignored.

If using the **csh**, the syntax on this man page does not apply, unless **/usr/bin/nice** is invoked at the beginning of the command line. Instead, refer to **csh**(1) for a different nice syntax, if in the **csh**.

SEE ALSO

csh(1), **nohup**(1), **priocntl**(1), **shell_builtins**(1), **nice**(2)

DIAGNOSTICS

/usr/bin/nice returns the exit status of *command*.

NOTES

An *increment* larger than 19 is equivalent to 19.

NAME

nis+, NIS+, nis – a new version of the network information name service

DESCRIPTION

NIS+ is a new version of the network information nameservice. This version differs in several significant ways from version 2, which is referred to as NIS or YP in earlier releases. Specific areas of enhancement include the ability to scale to larger networks, security, and the administration of the service.

The man pages for NIS+ are broken up into three basic categories. Those in section 1 are the user commands that are most often executed from a shell script or directly from the command line. Section 1M man pages describe utility commands that can be used by the network administrator to administer the service itself. The NIS+ programming API is described by man pages in section 3N.

All commands and functions that use NIS version 2 are prefixed by the letters **yp** as in **ypmatch**(1), **ypcat**(1), **yp_match**(3N), and **yp_first**(3N). Commands and functions that use the new replacement software NIS+ are prefixed by the letters **nis** as in **nismatch**(1), **nischown**(1), **nis_list**(3N), and **nis_add_entry**(3N). A complete list of NIS+ commands is in the **LIST OF COMMANDS** section.

This man page introduces the NIS+ terminology. It also describes the NIS+ namespace, authentication, and authorization policies.

NIS+ NAMESPACE

The naming model of NIS+ is based upon a tree structure. Each node in the tree corresponds to an NIS+ object. There are six types of NIS+ objects: *directory*, *table*, *group*, *link*, *entry*, and *private*.

NIS+ Directory Object

Each NIS+ namespace will have at least one NIS+ directory object. An NIS+ directory is like a UNIX file system directory which contains other NIS+ objects including NIS+ directories. The NIS+ directory that forms the root of the NIS+ namespace is called the root directory. There are two special NIS+ directories: **org_dir** and **groups_dir**. The **org_dir** directory consists of all the system-wide administration tables, such as **passwd**, **hosts**, and **mail_aliases**. The **groups_dir** directory consists of NIS+ group objects which are used for access control. The collection of **org_dir**, **groups_dir** and their parent directory is referred to as an NIS+ domain. NIS+ directories can be arranged in a tree-like structure so that the NIS+ namespace can match the organizational or administrative hierarchy.

NIS+ Table Object

NIS+ tables (not files), contained within NIS+ directories, store the actual information about some particular type. For example, the **hosts** system table stores information about the IP address of the hosts in that domain. NIS+ tables are multicolumn and the tables can be searched through any of the searchable columns. Each table object defines the schema for its table. The NIS+ tables consist of NIS+ entry objects. For each entry in the NIS+ table, there is an NIS+ entry object. NIS+ entry objects conform to the schema defined by the NIS+ table object.

NIS+ Group Object

NIS+ group objects are used for access control at group granularity. NIS+ group objects, contained within the **groups_dir** directory of a domain, contain a list of all the NIS+ principals within a certain NIS+ group. An NIS+ principal is a user or a machine making NIS+ requests.

NIS+ Link Object

NIS+ link objects are like UNIX symbolic file-system links—they are typically used for shortcuts in the NIS+ namespace.

Refer to **nis_objects**(3N) for more information about the NIS+ objects.

NIS+ NAMES

The NIS+ service defines two forms of names, *simple* names and *indexed* names. Simple names are used by the service to identify NIS+ objects contained within the NIS+ namespace. Indexed names are used to identify NIS+ entries contained within NIS+ tables. Furthermore, entries within NIS+ tables are returned to the caller as NIS+ objects of type *entry*. NIS+ objects are implemented as a union structure which is described in the file **<rpcsvc/nis_object.x>**. The differences between the various types and the meanings of the components of these objects are described in **nis_objects**(3N).

Simple Names

Simple names consist of a series of labels that are separated by the '.'(dot) character. Each label is composed of printable characters from the ISO Latin 1 set. Each label can be of any nonzero length, provided that the fully qualified name is fewer than **NIS_MAXNAMELEN** octets including the separating dots. (See **<rpcsvc/nis.h>** for the actual value of **NIS_MAXNAMELEN** in the current release.) Labels that contain special characters (see **Grammar**) must be quoted.

The NIS+ namespace is organized as a singly rooted tree. Simple names identify nodes within this tree. These names are constructed such that the leftmost label in a name identifies the leaf node and all of the labels to the right of the leaf identify that object's parent node. The parent node is referred to as the leaf's *directory*. This is a naming directory and should not be confused with a file system directory.

For example, the name *example.simple.name.* is a simple name with three labels, where *example* is the leaf node in this name, the directory of this leaf is *simple.name.* which by itself is a simple name. The leaf of which is *simple* and its directory is simply *name*.

The function **nis_leaf_of**(3N) returns the first label of a simple name. The function **nis_domain_of**(3N) returns the name of the directory that contains the leaf. Iterative use of these two functions can break a simple name into each of its label components.

The name '.' (dot) is reserved to name the *global root* of the namespace. For systems that are connected to the Internet, this global root will be served by a Domain Name Service. When an NIS+ server is serving a root directory whose name is not '.'(dot) this directory is referred to as a *local root*.

NIS+ names are said to be *fully qualified* when the name includes all of the labels identifying all of the directories, up to the global root. Names without the trailing dot are called *partially* qualified.

Indexed Names

Indexed names are compound names that are composed of a search criterion and a simple name. The search criterion component is used to select entries from a table; the simple name component is used to identify the NIS+ table that is to be searched. The search criterion is a series of column names and their desired values enclosed in bracket ‘**[ ]**’ characters. These criteria take the following form:

[*column_name***=***value***,** *column_name***=***value***, ...]**

A search criterion is combined with a simple name to form an indexed name by concatenating the two parts, separated by a ‘,’(comma) character as follows.

[*search-criterion* **],***table.directory*.

When multiple column name / value pairs are present in the search criterion, only those entries in the table that have the appropriate value in all columns specified are returned. When no column name / value pairs are specified in the search criterion, **[ ]**, *all* entries in the table are returned.

Grammar

The following text represents a context-free grammar that defines the set of legal NIS+ names. The terminals in this grammar are the characters ‘.’ (dot), ‘[’ (open bracket), ‘]’ (close bracket), ‘,’ (comma), ‘=’ (equals) and whitespace. Angle brackets (‘<’ and ‘>’), which delineate non-terminals, are not part of the grammar. The character ‘ | ’ (vertical bar) is used to separate alternate productions and should be read as ‘‘this production **OR** this production’’.

name ::= . | <*simple name*> | <*indexed name*>
simple name ::= <*string*>. | <*string*>.<*simple name*>
indexed name ::= <*search criterion*>,<*simple name*>
search criterion ::= [<*attribute list*>]
attribute list ::= <*attribute*> | <*attribute*>,<*attribute list*>
attribute ::= <*string*> = <*string*>
string ::= ISO Latin 1 character set except the character ’/’ (slash). The initial character may not be a terminal character or the characters ’@’ (at), ’+’ (plus), or (‘–’) hyphen.

Terminals that appear in strings must be quoted with ‘"’ (double quote). The ‘"’ character may be quoted by quoting it with itself ‘""’.

Name Expansion

The NIS+ service only accepts fully qualified names. However, since such names may be unwieldy, the NIS+ commands in section 1 employ a set of standard expansion rules that will attempt to fully qualify a partially qualified name. This expansion is actually done by the NIS+ library function **nis_getnames**(3N) which generates a list of names using the default NIS+ directory search path or the **NIS_PATH** environment variable. The default NIS+ directory search path includes all the names in its path. **nis_getnames( )** is invoked by the functions **nis_lookup**(3N) and **nis_list**(3N) when the **EXPAND_NAME** flag is used.

The **NIS_PATH** environment variable contains an ordered list of simple names. The names are separated by the ':' (colon) character. If any name in the list contains colons, the colon should be quoted as described in the **Grammar** section. When the list is exhausted, the resolution function returns the error **NIS_NOTFOUND**. This may mask the fact that the name existed but a server for it was unreachable. If the name presented to the list or lookup interface is fully qualified, the **EXPAND_NAME** flag is ignored.

In the list of names from the **NIS_PATH** environment variable, the '$' (dollar sign) character is treated specially. Simple names that end with the label '$' have this character replaced by the default directory (see **nis_local_directory**(3N)). Using "$" as a name in this list results in this name being replaced by the list of directories between the default directory and the global root that contain at least two labels.

Below is an example of this expansion. Given the default directory of *some.long.domain.name.*, and the **NIS_PATH** variable set to **fred.bar.:org_dir.$:$**. This path is initially broken up into the list:

1 **fred.bar.**

2 **org_dir.$**

3 **$**

The dollar sign in the second component is replaced by the default directory. The dollar sign in the third component is replaced with the names of the directories between the default directory and the global root that have at least two labels in them. The effective path value becomes:

1 **fred.bar.**

2a **org_dir.some.long.domain.name.**

3a **some.long.domain.name.**

3b **long.domain.name.**

3c **domain.name.**

Each of these simple names is appended to the partially qualified name that was passed to the **nis_lookup**(3N) or **nis_list**(3N) interface. Each is tried in turn until **NIS_SUCCESS** is returned or the list is exhausted.

If the NIS_PATH variable is not set, the path ‘‘$’’ is used.

The library function **nis_getnames**(3N) can be called from user programs to generate the list of names that would be attempted. The program **nisdefaults**(1) with the **–s** option can also be used to show the fully expanded path.

Concatenation Path

Normally all the entries for a certain type of information are stored within the table itself. However, there are times when it is desirable for the table to point to other tables where entries can be found. For example, you may want to store all the IP addresses in the host table for their own domain, and yet want to be able to resolve hosts in some other domain without explicitly specifying the new domain name. NIS+ provides a mechanism for concatenating different but related tables with a "NIS+ Concatenation Path". With a concatenation path, you can create a sort of flat namespace from a hierarchical structure. You can also create a table with no entries and just point the hosts or any other table to its parent domain. Note that with such a setup, you are moving the administrative burden of managing the tables to the parent domain. The concatenation path will slow down the request response time because more tables and more servers are searched. It will also decrease the availability if all the servers are incapacitated for a particular directory in the table path.

The NIS+ Concatenation Path is also referred to as the "table path". This path is set up at table creation time through **nistbladm**(1). You can specify more than one table to be concatenated and they will be searched in the given order. Note that the NIS+ client libraries, by default, will not follow the concatenation path set in site-specific tables. Refer to **nis_list**(3N) for more details.

Namespaces

The NIS+ service defines two additional *disjoint* namespaces for its own use. These namespaces are the NIS+ *Principal* namespace, and the NIS+ *Group* namespace. The names associated with the group and principal namespaces are syntactically identical to simple names. However, the information they represent *cannot* be obtained by directly presenting these names to the NIS+ interfaces. Instead, special interfaces are defined to map these names into NIS+ names so that they may then be resolved.

Principal Names

NIS+ principal names are used to uniquely identify users and machines that are making NIS+ requests. These names have the form:

principal.domain

Here *domain* is the fully qualified name of an NIS+ directory where the named principal's credentials can be found. See **Directories and Domains** for more information on domains. Note that in this name, *principal*, is not a leaf in the NIS+ namespace.

Credentials are used to map the identity of a host or user from one context such as a process UID into the NIS+ context. They are stored as records in an NIS+ table named *cred*, which always appears in the *org_dir* subdirectory of the directory named in the principal name.

This mapping can be expressed as a replacement function:

principal.domain –> **[cname**=*principal.domain* **],cred.org_dir**.*domain*

This latter name is an NIS+ name that can be presented to the **nis_list**(3N) interface for resolution. NIS+ principal names are administered using the **nisaddcred**(1M) command.

The *cred* table contains five columns named *cname*, *auth_name*, *auth_type*, *public_data*, and *private_data*. There is one record in this table for each identity mapping for an NIS+ principal. The current service supports two such mappings:

LOCAL	This mapping is used to map from the UID of a given process to the NIS+ principal name associated with that UID. If no mapping exists, the name *nobody* is returned. When the effective UID of the process is 0 (for example, the superuser), the NIS+ name associated with the host is returned. Note that UIDs are sensitive to the context of the machine on which the process is executing.
DES	This mapping is used to map to and from a Secure RPC ‘‘netname’’ into an NIS+ principal name. See **secure_rpc**(3N) for more information on netnames.

Note that since netnames contain the notion of a domain, they span NIS+ directories.

The NIS+ client library function **nis_local_principal**(3N) uses the *cred.org_dir* table to map the UNIX notion of an identity, a process' UID, into an NIS+ principal name. Shell programs can use the program **nisdefaults**(1) with the **-p** switch to return this information.

Mapping from UIDs to an NIS+ principal name is accomplished by constructing a query of the form:

[auth_type=LOCAL, auth_name=*uid***],cred.org_dir.***default-domain***.**

This query will return a record containing the NIS+ principal name associated with this UID, in the machine's default domain.

The NIS+ service uses the DES mapping to map the names associated with Secure RPC requests into NIS+ principal names. RPC requests that use Secure RPC include the *netname* of the client making the request in the RPC header. This netname has the form:

unix.*UID@domain*

The service constructs a query using this name of the form:

[auth_type=DES, auth_name=*netname***],cred.org_dir.***domain***.**

where the domain part is extracted from the netname rather than using the default domain. This query is used to look up the mapping of this netname into an NIS+ principal name in the domain where it was created.

This mechanism of mapping UID and netnames into an NIS+ principal name guarantees that a client of the NIS+ service has only one principal name. This principal name is used as the basis for authorization which is described below. All objects in the NIS+ namespace and all entries in NIS+ tables must have an owner specified for them. This owner field always contains an NIS+ principal name.

Group Names

Like NIS+ principal names, NIS+ group names take the form:
group_name**.domain**

All objects in the NIS+ namespace and all entries in NIS+ tables may optionally have a *group owner* specified for them. This group owner field, when filled in, always contains the fully qualified NIS+ group name.

The NIS+ client library defines several interfaces (**nis_groups**(3N)) for dealing with NIS+ groups. These interfaces internally map NIS+ group names into an NIS+ simple name which identifies the NIS+ group object associated with that group name. This mapping can be shown as follows:

group.domain –> *group***.groups_dir**.*domain*

This mapping eliminates collisions between NIS+ group names and NIS+ directory names. For example, without this mapping, a directory with the name *engineering.foo.com.*, would make it impossible to have a group named *engineering.foo.com.*. This is due to the restriction that within the NIS+ namespace, a name unambiguously identifies a single object. With this mapping, the NIS+ *group* name *engineering.foo.com.* maps to the NIS+ *object* name *engineering.groups_dir.foo.com.*

The contents of a group object is a list of NIS+ principal names, and the names of other NIS+ groups. See **nis_groups**(3N) for a more complete description of their use.

NIS+ SECURITY

NIS+ defines a security model to control access to information managed by the service. The service defines access rights that are selectively granted to individual clients or groups of clients. Principal names and group names are used to define clients and groups of clients that may be granted or denied access to NIS+ information. These principals and groups are associated with NIS+ domains as defined below.

The security model also uses the notion of a class of principals called *nobody*, which contains all clients, whether or not they have authenticated themselves to the service. The class *world* includes any client who has been authenticated.

Directories and Domains

Some directories within the NIS+ namespace are referred to as NIS+ *Domains*. Domains are those NIS+ directories that contain the subdirectories *groups_dir* and *org_dir*. Further, the subdirectory *org_dir* should contain the table named *cred*. NIS+ Group names and NIS+ Principal names **always** include the NIS+ domain name after their first label.

Authentication

The NIS+ name service uses Secure RPC for the integrity of the NIS+ service. This requires that users of the service and their machines must have a Secure RPC key pair associated with them. This key is initially generated with either the **nisaddcred**(1M) or **nisclient**(1M) commands and modified with the **chkey**(1) or **nispasswd**(1) commands.

The use of Secure RPC allows private information to be stored in the name service that will not be available to untrusted machines or users on the network.

In addition to the Secure RPC key, users need a mapping of their UID into an NIS+ principal name. This mapping is created by the system administrator using the **nisclient**(1M) or **nisaddcred**(1M) command.

Users that will be using machines in several NIS+ domains must insure that they have a *local* credential entry in each of those domains. This credential should be created with the NIS+ principal name of the user in their ‘‘home’’ domain. For the purposes of NIS+ and Secure RPC, the home domain is defined to be the one where your Secure RPC key pair is located.

Authorization

The NIS+ service defines four access rights that can be granted or denied to clients of the service. These rights are *read, modify, create*, and *destroy*. These rights are specified in the object structure at creation time and may be modified later with the **nischmod**(1) command. In general, the rights granted for an object apply only to that object. However, for purposes of authorization, rights granted to clients reading *directory* and *table* objects are granted to those clients for all of the objects ‘‘contained’’ by the parent object. This notion of containment is abstract. The objects do not actually contain other objects within them. Note that *group* objects do contain the list of principals within their definition.

Access rights are interpreted as follows:

read	This right grants read access to an object. For directory and table objects, having read access on the parent object conveys read access to all of the objects that are direct children of a directory, or entries within a table.

modify	This right grants modification access to an existing object. Read access is not required for modification. However, in many applications, one will need to read an object before modifying it. Such modify operations will fail unless read access is also granted.
create	This right gives a client permission to create new objects where one had not previously existed. It is only used in conjunction with directory and table objects. Having create access for a table allows a client to add additional entries to the table. Having create access for a directory allows a client to add new objects to an NIS+ directory.
destroy	This right gives a client permission to destroy or remove an existing object or entry. When a client attempts to destroy an entry or object by removing it, the service first checks to see if the table or directory containing that object grants the client destroy access. If it does, the operation proceeds, if the containing object does not grant this right then the object itself is checked to see if it grants this right to the client. If the object grants the right, then the operation proceeds; otherwise the request is rejected.

Each of these rights may be granted to any one of four different categories.

owner	A right may be granted to the *owner* of an object. The owner is the NIS+ principal identified in the owner field. The owner can be changed with the **nischown**(1) command. Note that if the owner does not have modification access rights to the object, the owner cannot change any access rights to the object, unless the owner has modification access rights to its parent object.
group owner	A right may be granted to the *group owner* of an object. This grants the right to any principal that is identified as a member of the group associated with the object. The group owner may be changed with the **nischgrp**(1) command. The object owner need not be a member of this group.
world	A right may be granted to everyone in the *world*. This grants the right to all clients who have authenticated themselves with the service.
nobody	A right may be granted to the *nobody* principal. This has the effect of granting the right to any client that makes a request of the service, regardless of whether they are authenticated or not.

Note that for bootstrapping reasons, directory objects that are NIS+ domains, the *org_dir* subdirectory and the *cred* table within that subdirectory must have *read* access to the *nobody* principal. This makes navigation of the namespace possible when a client is in the process of locating its credentials. Granting this access does not allow the contents of other tables within *org_dir* to be read (such as the entries in the password table) unless the table itself gives "real" access rights to the *nobody* principal.

Directory Authorization

Additional capabilities are provided for granting access rights to clients for directories. These rights are contained within the *object access rights* (OAR) structure of the directory. This structure allows the NIS+ service to grant rights that are not granted by the directory object to be granted for objects contained by the directory of a specific type.

An example of this capability is a directory object which does not grant create access to all clients, but does grant create access in the OAR structure for *group* type objects to clients who are members of the NIS+ group associated with the directory. In this example the only objects that could be created as children of the directory would have to be of the type *group*.

Another example is a directory object that grants create access only to the owner of the directory, and then additionally grants create access through the OAR structure for objects of type *table*, *link*, *group*, and *private* to any member of the directory's group. This has the effect of giving nearly complete create access to the group with the exception of creating subdirectories. This restricts the creation of new NIS+ domains because creating a domain requires creating both a *groups_dir* and *org_dir* subdirectory.

Note that there is currently no command line interface to set or change the OAR of the directory object.

Table Authorization

As with directories, additional capabilities are provided for granting access to entries within tables. Rights granted to a client by the access rights field in a table object apply to the table object and all of the entry objects ‘‘contained’’ by that table. If an access right is not granted by the table object, it may be granted by an entry within the table. This holds for all rights except *create*.

For example, a table may not grant read access to a client performing a **nis_list**(3N) operation on the table. However, the access rights field of entries within that table may grant read access to the client. Note that access rights in an entry are granted to the owner and group owner of the *entry* and not the owner or group of the table. When the list operation is performed, all entries that the client has read access to are returned. Those entries that do not grant read access are not returned. If none of the entries that match the search criterion grant read access to the client making the request, no entries are returned and the result status contains the NIS_NOTFOUND error code.

Access rights that are granted by the rights field in an entry are granted for the entire entry. However, in the table object an additional set of access rights is maintained for each column in the table. These rights apply to the equivalent column in the entry. The rights are used to grant access when neither the table nor the entry itself grant access. The access rights in a column specification apply to the owner and group owner of the entry rather than the owner and group owner of the table object.

When a read operation is performed, if read access is not granted by the table and is not granted by the entry but *is* granted by the access rights in a column, that entry is returned with the correct values in all columns that are readable and the string *∗***NP***∗* (No Permission) in columns where read access is not granted.

As an example, consider a client that has performed a list operation on a table that does not grant read access to that client. Each entry object that satisfied the search criterion specified by the client is examined to see if it grants read access to the client. If it does, it is included in the returned result. If it does not, then each column is checked to see if it grants read access to the client. If any columns grant read access to the client, data in those columns is returned. Columns that do not grant read access have their contents replaced by the string ∗**NP**∗. If none of the columns grant read access, then the entry is not returned.

LIST OF COMMANDS

The following lists all commands and programming functions related to NIS+:

NIS+ User Commands

nisaddent(1)	add /etc files and NIS maps into their corresponding NIS+ tables
niscat(1)	display NIS+ tables and objects
nischgrp(1)	change the group owner of a NIS+ object
nischmod(1)	change access rights on a NIS+ object
nischown(1)	change the owner of a NIS+ object
nischttl(1)	change the time to live value of a NIS+ object
nisdefaults(1)	display NIS+ default values
niserror(1)	display NIS+ error messages
nisgrep(1)	utilities for searching NIS+ tables
nisgrpadm(1)	NIS+ group administration command
nisln(1)	symbolically link NIS+ objects
nisls(1)	list the contents of a NIS+ directory
nismatch(1)	utilities for searching NIS+ tables
nismkdir(1)	create NIS+ directories
nispasswd(1)	change NIS+ password information
nisrm(1)	remove NIS+ objects from the namespace
nisrmdir(1)	remove NIS+ directories
nisshowcache(1)	NIS+ utility to print out the contents of the shared cache file
nistbladm(1)	NIS+ table administration command
nistest(1)	return the state of the NIS+ namespace using a conditional expression

NIS+ Administrative Commands

aliasadm(1M)	manipulate the NIS+ aliases map
nis_cachemgr(1M)	NIS+ utility to cache location information about NIS+ servers
nisaddcred(1M)	create NIS+ credentials
nisaddent(1M)	create NIS+ tables from corresponding /etc files or NIS maps
nisclient(1M)	initialize NIS+ credentials for NIS+ principals
nisd(1M)	NIS+ service daemon
nisd_resolv(1M)	NIS+ service daemon
nisinit(1M)	NIS+ client and server initialization utility
nislog(1M)	display the contents of the NIS+ transaction log
nisping(1M)	send ping to NIS+ servers
nispopulate(1M)	populate the NIS+ tables in a NIS+ domain
nisserver(1M)	set up NIS+ servers
nissetup(1M)	initialize a NIS+ domain
nisshowcache(1M)	NIS+ utility to print out the contents of the shared cache file

nisstat(1M)	report NIS+ server statistics
nisupdkeys(1M)	update the public keys in a NIS+ directory object
rpc.nisd(1M)	NIS+ service daemon
rpc.nisd_resolv(1M)	NIS+ service daemon
sysidnis(1M)	system configuration

NIS+ Programming API

__nis_map_group(3N)	NIS+ group manipulation functions
db_add_entry(3N)	NIS+ Database access functions
db_checkpoint(3N)	NIS+ Database access functions
db_create_table(3N)	NIS+ Database access functions
db_destroy_table(3N)	NIS+ Database access functions
db_first_entry(3N)	NIS+ Database access functions
db_free_result(3N)	NIS+ Database access functions
db_initialize(3N)	NIS+ Database access functions
db_list_entries(3N)	NIS+ Database access functions
db_next_entry(3N)	NIS+ Database access functions
db_remove_entry(3N)	NIS+ Database access functions
db_reset_next_entry(3N)	NIS+ Database access functions
db_standby(3N)	NIS+ Database access functions
db_table_exists(3N)	NIS+ Database access functions
db_unload_table(3N)	NIS+ Database access functions
nis_add(3N)	NIS+ namespace functions
nis_add_entry(3N)	NIS+ table functions
nis_addmember(3N)	NIS+ group manipulation functions
nis_checkpoint(3N)	misc NIS+ log administration functions
nis_clone_object(3N)	NIS+ subroutines
nis_creategroup(3N)	NIS+ group manipulation functions
nis_db(3N)	NIS+ Database access functions
nis_destroy_object(3N)	NIS+ subroutines
nis_destroygroup(3N)	NIS+ group manipulation functions
nis_dir_cmp(3N)	NIS+ subroutines
nis_domain_of(3N)	NIS+ subroutines
nis_error(3N)	display NIS+ error messages
nis_first_entry(3N)	NIS+ table functions
nis_freenames(3N)	NIS+ subroutines
nis_freeresult(3N)	NIS+ namespace functions
nis_freeservlist(3N)	miscellaneous NIS+ functions
nis_freetags(3N)	miscellaneous NIS+ functions
nis_getnames(3N)	NIS+ subroutines
nis_getservlist(3N)	miscellaneous NIS+ functions
nis_groups(3N)	NIS+ group manipulation functions
nis_ismember(3N)	NIS+ group manipulation functions
nis_leaf_of(3N)	NIS+ subroutines
nis_lerror(3N)	display some NIS+ error messages
nis_list(3N)	NIS+ table functions

nis_local_directory(3N)	NIS+ local names
nis_local_group(3N)	NIS+ local names
nis_local_host(3N)	NIS+ local names
nis_local_names(3N)	NIS+ local names
nis_local_principal(3N)	NIS+ local names
nis_lookup(3N)	NIS+ namespace functions
nis_map_group(3N)	NIS+ group manipulation functions
nis_mkdir(3N)	miscellaneous NIS+ functions
nis_modify(3N)	NIS+ namespace functions
nis_modify_entry(3N)	NIS+ table functions
nis_name_of(3N)	NIS+ subroutines
nis_names(3N)	NIS+ namespace functions
nis_next_entry(3N)	NIS+ table functions
nis_objects(3N)	NIS+ object formats
nis_perror(3N)	display NIS+ error messages
nis_ping(3N)	misc NIS+ log administration functions
nis_print_group_entry(3N)	NIS+ group manipulation functions
nis_print_object(3N)	NIS+ subroutines
nis_remove(3N)	NIS+ namespace functions
nis_remove_entry(3N)	NIS+ table functions
nis_removemember(3N)	NIS+ group manipulation functions
nis_rmdir(3N)	miscellaneous NIS+ functions
nis_server(3N)	miscellaneous NIS+ functions
nis_servstate(3N)	miscellaneous NIS+ functions
nis_sperrno(3N)	display NIS+ error messages
nis_sperror(3N)	display NIS+ error messages
nis_sperror_r(3N)	display NIS+ error messages
nis_stats(3N)	miscellaneous NIS+ functions
nis_subr(3N)	NIS+ subroutines
nis_tables(3N)	NIS+ table functions
nis_verifygroup(3N)	NIS+ group manipulation functions

NIS+ Files and Directories

nisfiles(4)	NIS+ database files and directory structure

FILES

<rpcsvc/nis_object.x>	protocol description of an NIS+ object
<rpcsvc/nis.x>	defines the NIS+ protocol using the RPC language as described in the *Network Interfaces Programmer's Guide*
<rpcsvc/nis.h>	should be included by all clients of the NIS+ service

SEE ALSO

nischown(1), **nisdefaults**(1), **nismatch**(1), **nispasswd**(1), **admintool**(1M), **newkey**(1M), **nisaddcred**(1M), **nisclient**(1M), **nispopulate**(1M), **nisserver**(1M), **nis_add_entry**(3N), **nis_domain_of**(3N), **nis_getnames**(3N), **nis_groups**(3N), **nis_leaf_of**(3N), **nis_list**(3N), **nis_local_directory**(3N), **nis_lookup**(3N), **nis_objects**(3N)

Network Interfaces Programmer's Guide

Describes the application programming interfaces for networks including NIS+

Name Service Configuration Guide

Describes how to plan for and configure an NIS+ namespace

Name Services Administration Guide

Describes how to administer a running NIS+ namespace and modify its security

NIS+ Transition Guide

Describes how to make the transition from NIS to NIS+

Administration Application Reference Manual

Describes the **admintool**(1M) window interface for modifying the data in NIS+ tables

NAME

niscat – display NIS+ tables and objects

SYNOPSIS

niscat [–**AhLMv**] *tablename* . . .

niscat [–**ALMP**] –**o** *name* . . .

AVAILABILITY

SUNWnisu

DESCRIPTION

In the first synopsis, **niscat** displays the contents of the NIS+ tables named by *tablename*. In the second synopsis, it displays the internal representation of the NIS+ objects named by *name*.

OPTIONS

–**A**	Display the data within the table and all of the data in tables in the initial table's concatenation path.
–**h**	Display the header line prior to displaying the table. The header consists of the '#' (hash) character followed by the name of each column. The column names are separated by the table separator character.
–**L**	Follow links. When this option is specified, if *tablename* or *name* names a LINK type object, the link is followed and the object or table named by the link is displayed.
–**M**	Master server only. This option specifies that the request should be sent to the master server of the named data. This guarantees that the most up-to-date information is seen at the possible expense of increasing the load on the master server and increasing the possibility of the NIS+ server being unavailable or busy for updates.
–**P**	Follow concatenation path. This option specifies that the request should follow the concatenation path of a table if the initial search is unsuccessful. This option is only useful when using an indexed name for *name* and the –**o** option.
–**v**	Display binary data directly. This option displays columns containing binary data on the standard output. Without this option binary data is displayed as the string ***BINARY***.
–**o** *name*	Display the internal representation of the named NIS+ object(s). If *name* is an indexed name (see **nismatch**(1)), then each of the matching entry objects is displayed. This option is used to display access rights and other attributes of individual columns.

EXAMPLES

This example displays the contents of the hosts table.

```
example% niscat –h host.org_dir
# cname   name      addr              comment
client1   client1   129.144.201.100   Joe Smith
crunchy   crunchy   129.144.201.44    Jane Smith
crunchy   softy     129.144.201.44
```

The string ∗**NP**∗ is returned in those fields where the user has insufficient access rights.

Display the **passwd.org_dir** on the standard output.

```
example% niscat passwd.org_dir
```

Display the contents of table **frodo** and the contents of all tables in its concatenation path.

```
example% niscat –A frodo
```

Display the entries in the table **groups.org_dir** as NIS+ objects. Note that the brackets are protected from the shell by single quotes.

```
example% niscat –o ’[ ]groups.org_dir’
```

Display the table object of the **passwd.org_dir** table.

```
example% niscat –o passwd.org_dir
```

The previous example displays the passwd table object and not the passwd table. The table object include information such as the number of columns, column type, searchable or not searchable separator, access rights, and other defaults.

Display the directory object for **org_dir**, which includes information such as the access rights and replica information.

```
example% niscat –o org_dir
```

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ table name is not fully qualified, each directory specified will be searched until the table is found (see **nisdefaults**(1)).

EXIT CODES

niscat returns **0** on success and **1** on failure.

SEE ALSO

nis+(1), **nismatch**(1), **nistbladm**(1), **nisdefaults**(1), **nis_objects**(3N), **nis_tables**(3N)

NOTES

Columns without values in the table are displayed by two adjacent separator characters.

NAME

nischgrp – change the group owner of a NIS+ object

SYNOPSIS

nischgrp [–**AfLP**] *group name* . . .

AVAILABILITY

SUNWnisu

DESCRIPTION

nischgrp changes the group owner of the NIS+ objects or entries specified by *name* to the specified NIS+ *group*. Entries are specified using indexed names (see **nismatch**(1)). If *group* is not a fully qualified NIS+ group name, it will be resolved using the directory search path (see **nisdefaults**(1)).

The only restriction on changing an object's group owner is that you must have modify permissions for the object.

This command will fail if the master NIS+ server is not running.

OPTIONS

–**A**	Modify all entries in all tables in the concatenation path that match the search criterion specified in *name*. This option implies the –**P** switch.
–**f**	Force the operation and fail silently if it does not succeed.
–**L**	Follow links and change the group owner of the linked object or entries rather than the group owner of the link itself.
–**P**	Follow the concatenation path within a named table. This option only makes sense when either *name* is an indexed name or the –**L** switch is also specified and the named object is a link pointing to entries.

EXAMPLES

The following two examples show how to change the group owner of an object to a group in a different domain, and how to change it to a group in the local domain, respectively.

```
example% nischgrp  newgroup.remote.domain.  object
example% nischgrp  my-buds  object
```

This example shows how to change the group owner for a password entry.

```
example% nischgrp  admins  '[uid=99],passwd.org_dir'
```

In the previous example, **admins** is a NIS+ group in the same domain.

The next two examples change the group owner of the object or entries pointed to by a link, and the group owner of all entries in the **hobbies** table.

```
example% nischgrp  –L  my-buds  linkname
example% nischgrp  my-buds  '[],hobbies'
```

ENVIRONMENT

NIS_PATH	If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

EXIT CODES

nischgrp returns **0** on success and **1** on failure.

SEE ALSO

nis+(1), **nischmod**(1), **nischown**(1), **nisdefaults**(1), **nisgrpadm**(1), **nis_objects**(3N)

NOTES

The NIS+ server will check the validity of the group name prior to effecting the modification.

NAME

nischmod – change access rights on a NIS+ object

SYNOPSIS

nischmod [**–AfLP**] *mode name* . . .

AVAILABILITY

SUNWnisu

DESCRIPTION

nischmod changes the access rights (mode) of the NIS+ objects or entries specified by *name* to *mode*. Entries are specified using indexed names (see **nismatch**(1)). Only principals with modify access to an object may change its mode.

mode has the following form:

> *rights* [, *rights*] . . .

rights has the form:

> [*who*] *op permission* [*op permission*] . . .

who is a combination of:

n	Nobody's permissions.
o	Owner's permissions.
g	Group's permissions.
w	World's permissions.
a	All, or **owg**.

If *who* is omitted, the default is **a**.

op is one of:

+	To grant the *permission*.
–	To revoke the *permission*.
=	To set the permissions explicitly.

permission is any combination of:

r	Read.
m	Modify.
c	Create.
d	Destroy.

OPTIONS

–A	Modify all entries in all tables in the concatenation path that match the search criteria specified in *name*. This option implies the **–P** switch.
–f	Force the operation and fail silently if it does not succeed.
–L	Follow links and change the permission of the linked object or entries rather than the permission of the link itself.
–P	Follow the concatenation path within a named table. This option is only applicable when either *name* is an indexed name or the **–L** switch is also specified and the named object is a link pointing to an entry.

EXAMPLES

This example gives everyone read access to an object. (i.e., access for owner, group, and all).

```
example% nischmod a+r object
```

This example denies create and modify privileges to **group** and unauthenticated clients (**nobody**).

```
example% nischmod gn–cm object
```

In this example, a complex set of permissions are set for an object.

```
example% nischmod o=rmcd,g=rm,w=rc,n=r object
```

This example sets the permissions of an entry in the password table so that the group owner can modify them.

```
example% nischmod g+m '[uid=55],passwd.org_dir'
```

The next example changes the permissions of a linked object.

```
example% nischmod –L w+mr linkname
```

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

EXIT CODES

nischmod returns **0** on success and **1** on failure.

SEE ALSO

chmod(1), **nis+**(1), **nischgrp**(1), **nischown**(1), **nisdefaults**(1), **nis_objects**(3N)

NOTES

Unlike the system **chmod**(1) command, this command does not accept an octal notation.

NAME

nischown – change the owner of a NIS+ object

SYNOPSIS

nischown [–**AfLP**] *owner name* . . .

AVAILABILITY

SUNWnisu

DESCRIPTION

nischown changes the owner of the NIS+ objects or entries specified by *name* to *owner*. Entries are specified using indexed names (see **nismatch**(1)). If *owner* is not a fully qualified NIS+ principal name (see **nisaddcred**(1M)), the default domain (see **nisdefaults**(1)) will be appended to it.

The only restriction on changing an object's owner is that you must have modify permissions for the object. Note: If you are the current owner of an object and you change ownership, you may not be able to regain ownership unless you have modify access to the new object.

The command will fail if the master NIS+ server is not running.

OPTIONS

–**A**	Modify all entries in all tables in the concatenation path that match the search criteria specified in *name*. It implies the –**P** option.
–**f**	Force the operation and fail silently if it does not succeed.
–**L**	Follow links and change the owner of the linked object or entries rather than the owner of the link itself.
–**P**	Follow the concatenation path within a named table. This option is only meaningful when either *name* is an indexed name or the –**L** option is also specified and the named object is a link pointing to entries.

EXAMPLES

The following two examples show how to change the owner of an object to a principal in a different domain, and to change it to a principal in the local domain, respectively.

```
example% nischown bob.remote.domain. object
example% nischown skippy object
```

The next example shows how to change the owner of an entry in the passwd table.

```
example% nischown bob.remote.domain. '[uid=99],passwd.org_dir'
```

This example shows how to change the object or entries pointed to by a link.

```
example% nischown –L skippy linkname
```

ENVIRONMENT

NIS_PATH	If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

EXIT CODES

nischown returns **0** on success and **1** on failure.

SEE ALSO

nis+(1), **nischgrp**(1), **nischmod**(1), **nischttl**(1), **nisdefaults**(1), **nisaddcred**(1M), **nis_objects**(3N)

NOTES

The NIS+ server will check the validity of the name before making the modification.

NAME

nischttl – change the time to live value of a NIS+ object

SYNOPSIS

nischttl [–**AfLP**] *time name* . . .

AVAILABILITY

SUNWnisu

DESCRIPTION

nischttl changes the time to live value (**ttl**) of the NIS+ objects or entries specified by *name* to *time*. Entries are specified using indexed names (see **nismatch**(1)).

The time to live value is used by object caches to expire objects within their cache. When an object is read into the cache, this value is added to the current time in seconds yielding the time when the cached object would expire. The object may be returned from the cache until the current time is earlier than the calculated expiration time. When the expiration time has been reached, the object will be flushed from the cache.

The time to live *time* may be specified in seconds or in days, hours, minutes, seconds format. The latter format uses a suffix letter of **d**, **h**, **m**, or **s** to identify the units of time. See the examples below for usage.

The command will fail if the master NIS+ server is not running.

OPTIONS

–**A**	Modify all tables in the concatenation path that match the search criterion specified in *name*. This option implies the –**P** switch.
–**f**	Force the operation and fail silently if it does not succeed.
–**L**	Follow links and change the time to live of the linked object or entries rather than the time to live of the link itself.
–**P**	Follow the concatenation path within a named table. This option only makes sense when either *name* is an indexed name or the –**L** switch is also specified and the named object is a link pointing to entries.

EXAMPLES

The following example shows how to change the **ttl** of an object using the seconds format and the days, hours, minutes, seconds format. The **ttl** of the second object is set to 1 day and 12 hours.

```
example% nischttl 184000  object
example% nischttl 1d12h object
```

This example shows how to change the **ttl** for a password entry.

```
example% nischttl 1h30m '[uid=99],passwd.org_dir'
```

The next two examples change the **ttl** of the object or entries pointed to by a link, and the **ttl** of all entries in the **hobbies** table.

```
example% nischttl –L 12h linkname
example% nischttl 3600 '[],hobbies
```

ENVIRONMENT

NIS_PATH	If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

EXIT CODES

nischttl returns **0** on success and **1** on failure.

SEE ALSO

nis+(1), **nischgrp**(1), **nischmod**(1), **nischown**(1), **nisdefaults**(1), **nis_objects**(3N)

NOTES

Setting a high **ttl** value allows objects to stay persistent in caches for a longer period of time and can improve performance. However, when an object changes, in the worst case, the number of seconds in this attribute must pass before that change is visible to all clients. Setting a **ttl** value of **0** means that the object should not be cached at all.

A high **ttl** value is a week, a low value is less than a minute. Password entries should have **ttl** values of about 12 hours (easily allows one password change per day), entries in the RPC table can have **ttl** values of several weeks (this information is effectively unchanging).

Only directory and group objects are cached in this implementation.

NAME

nisdefaults – display NIS+ default values

SYNOPSIS

nisdefaults [**–adghprstv**]

AVAILABILITY

SUNWnisu

DESCRIPTION

nisdefaults prints the default values that are returned by calls to the NIS+ local name functions (see **nis_local_names**(3N)). With no options specified, all defaults will be printed in a verbose format. With options, only that option is displayed in a terse form suitable for shell scripts. See the example below.

OPTIONS

–a	Print all defaults in a terse format.
–d	Print the default domain name.
–g	Print the default group name.
–h	Print the default host name.
–p	Print the default principal name.
–r	Print the default access rights with which new objects will be created.
–s	Print the default directory search path.
–t	Print the default time to live value.
–v	Print the defaults in a verbose format. This prepends an identifying string to the output.

EXAMPLES

The following prints the NIS+ defaults for a root process on machine **example** in the **foo.bar.** domain.

```
example# nisdefaults
Principal Name : example.foo.bar.
Domain Name    : foo.bar.
Host Name      : example.foo.bar.
Group Name     :
Access Rights  : ----rmcdr---r---
Time to live   : 12:00:00
Search Path    : foo.bar.
```

This example sets a variable in a shell script to the default domain.

```
DOMAIN=`nisdefaults -d`
```

This example prints out the default time to live in a verbose format.

```
example% nisdefaults -tv
Time to live   : 12:00:00
```

This example prints out the time to live in the terse format:

```
example% nisdefaults -t
43200
```

ENVIRONMENT

Several environment variables affect the defaults associated with a process.

NIS_DEFAULTS This variable contains a defaults string that will override the NIS+ standard defaults. The defaults string is a series of tokens separated by colons. These tokens represent the default values to be used for the generic object properties. All of the legal tokens are described below.

ttl=*time*
This token sets the default time to live for objects that are created. The value *time* is specified in the format as defined by the **nischttl**(1) command. The default value is **12** hours.

owner=*ownername*
This token specifies that the NIS+ principal *ownername* should own created objects. The default for this value is the principal who is executing the command.

group=*groupname*
This token specifies that the group *groupname* should be the group owner for created objects. The default is **NULL**.

access=*rights*
This token specifies the set of access rights that are to be granted for created objects. The value *rights* is specified in the format as defined by the **nischmod**(1) command. The default value is −−−−**rmcdr**−−−**r**−−−.

NIS_GROUP This variable contains the name of the local NIS+ group. If the name is not fully qualified, the default domain will be appended to it.

NIS_PATH This variable overrides the default NIS+ directory search path. It contains an ordered list of directories separated by ':' (colon) characters. The '$' (dollar sign) character is treated specially. Directory names that end in '$' have the default domain appended to them, and a '$' by itself is replaced by the list of directories between the default domain and the global root that are at least two levels deep. The default NIS+ directory search path is '$'.

Refer to the **Name Expansion** subsection in **nis+**(1) for more details.

SEE ALSO

nis+(1), **nis_local_names**(3N)

NAME

niserror – display NIS+ error messages

SYNOPSIS

niserror *error-num*

AVAILABILITY

SUNWnisu

DESCRIPTION

niserror prints the NIS+ error associated with status value *error-num* on the standard output. It is used by shell scripts to translate NIS+ error numbers that are returned into text messages.

EXAMPLES

The following example prints the error associated with the error number **20**:

```
example% niserror 20
Not Found, no such name
```

SEE ALSO

nis+(1), **nis_error**(3N)

NAME

nisgrpadm – NIS+ group administration command

SYNOPSIS

nisgrpadm –a | **–r** | **–t**] [**–s**] *group principal*. . .

nisgrpadm –c | **–d** | **–l** [**–M**] [**–s**] *group*

AVAILABILITY

SUNWnisu

DESCRIPTION

nisgrpadm is used to administer NIS+ groups. This command administers both groups and the groups' membership lists. **nisgrpadm** can create, destroy, or list NIS+ groups. **nisgrpadm** can be used to administer a group's membership list. It can add or delete principals to the group, or test principals for membership in the group.

The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.

There are three types of group members:

- An *explicit* member is just a NIS+ principal-name, for example "wickedwitch.west.oz."
- An *implicit* ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are *not* included.
- A *recursive* ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here.

Any member may be made *negative* by prefixing it with a minus sign ('–'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.

A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.

OPTIONS

–a Add the list of NIS+ principals specified to *group*. The principal name should be fully qualified.

–c Create *group* in the NIS+ namespace. The NIS+ group name should be fully qualified.

–d Destroy (remove) *group* from the namespace.

–l List the membership list of the specified *group*. (See **–M**.)

–M Master server only. Send the lookup to the master server of the named data. This guarantees that the most up to date information is seen at the possible expense that the master server may be busy. Note that the **–M** flag is applicable only with the **–l** flag.

–r Remove the list of principals specified from *group*. The principal name should be fully qualified.

–s Work silently. Results are returned using the exit status of the command. This status can be translated into a text string using the **niserror**(1) command.

–t Display whether the principals specified are members in *group*.

EXAMPLES

Administering Groups

This example shows how to create a group in the **foo.com.** domain.

```
example% nisgrpadm –c my_buds.foo.com.
```

This example shows how to remove the group from the current domain.

```
example% nisgrpadm –d freds_group
```

Administering Members

This example shows how one would add two principals, **bob** and **betty** to the group **my_buds.foo.com.**

```
example% nisgrpadm –a my_buds.foo.com. bob.bar.com. betty.foo.com.
```

This example shows how to remove **betty** from **freds_group**.

```
example% nisgrpadm –r freds_group betty.foo.com.
```

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ group name is not fully qualified, each directory specified will be searched until the group is found (see **nisdefaults**(1)).

SEE ALSO

nis+(1), **nischgrp**(1), **nisdefaults**(1), **niserror**(1), **nis_groups**(3N)

DIAGNOSTICS

NIS_SUCCESS	On success, this command returns an exit status of **0**.
NIS_PERMISSION	When you do not have the needed access right to change the group, the command returns this error.
NIS_NOTFOUND	This is returned when the group does not exist.
NIS_TRYAGAIN	This error is returned when the server for the group's domain is currently checkpointing or otherwise in a read-only state. The command should be retried at a later date.
NIS_MODERROR	This error is returned when the group was modified by someone else during the execution of the command. Reissue the command and optionally recheck the group's membership list.

NOTES

Principal names *must* be fully qualified, whereas groups can be abbreviated on all operations *except* create.

NAME

nisln – symbolically link NIS+ objects

SYNOPSIS

nisln [–**L**] [–**D** *defaults*] *name linkname*

AVAILABILITY

SUNWnisu

DESCRIPTION

The **nisln** command links a NIS+ object named *name* to a NIS+ name *linkname.* If *name* is an indexed name (see **nismatch**(1)), the link points to entries within a NIS+ table. Clients wishing to look up information in the name service can use the **FOLLOW_LINKS** flag to force the client library to follow links to the name they point to. Further, all of the NIS+ administration commands accept the –**L** switch indicating they should follow links (see **nis_names**(3N) for a description of the **FOLLOW_LINKS** flag).

OPTIONS

–**L** When present, this option specifies that this command should follow links. If *name* is itself a link, then this command will follow it to the linked object that it points to. The new link will point to that linked object rather than to *name*.

–**D** *defaults* Specify a different set of defaults to be used for the creation of the link object. The *defaults* string is a series of tokens separated by colons. These tokens represent the default values to be used for the generic object properties. All of the legal tokens are described below.

ttl=*time*	This token sets the default time to live for objects that are created by this command. The value *time* is specified in the format as defined by the **nischttl**(1) command. The default is **12** hours.
owner=*ownername*	This token specifies that the NIS+ principal *ownername* should own the created object. The default for this value is the the principal who is executing the command.
group=*groupname*	This token specifies that the group *groupname* should be the group owner for the object that is created. The default is **NULL**.
access=*rights*	This token specifies the set of access rights that are to be granted for the given object. The value *rights* is specified in the format as defined by the **nischmod**(1) command. The default value is −−−−**rmcdr**−−−**r**−−−.

EXAMPLES

In this example we create a link in the domain **foo.com.** named **hosts** that points to the object **hosts.bar.com**.

```
example% nisln hosts.bar.com. hosts.foo.com.
```

In this example we make a link *example.sun.com.* that points to an entry in the hosts table in *eng.sun.com.*

```
example% nisln '[name=example],hosts.eng.sun.com.' example.sun.com.
```

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

EXIT CODES .LP

nisln returns **0** on success and **1** on failure.

SEE ALSO

nisdefaults(1), **nismatch**(1), **nisrm**(1), **nistbladm**(1), **nis_names**(3N), **nis_tables**(3N)

NOTES

When creating the link, **nisln** verifies that the linked object exists. Once created, the linked object may be deleted or replaced and the link will not be affected. At that time the link will become invalid and attempts to follow it will return **NIS_LINKNAMEERROR** to the client. When the path attribute in tables specifies a link rather than another table, the link will be followed if the flag **FOLLOW_LINKS** was present in the call to **nis_list( )** (see **nis_tables**(3N)) and ignored if the flag is not present. If the flag is present and the link is no longer valid, a warning is sent to the system logger and the link is ignored.

NAME

nisls – list the contents of a NIS+ directory

SYNOPSIS

nisls [**–dglLmMR**] [*name* . . .]

AVAILABILITY

SUNWnisu

DESCRIPTION

For each *name* that is a NIS+ directory, **nisls** lists the contents of the directory. For each *name* that is a NIS+ object other than a directory, **nisls** simply echos the name. If no *name* is specified, the first directory in the search path (see **nisdefaults**(1)) is listed.

OPTIONS

–d Treat NIS+ directories like other NIS+ objects, rather than listing their contents.

–g Display group owner instead of owner when listing in long format.

–l List in long format. This option displays additional information about the objects such as their type, creation time, owner, and access rights.

The access rights are listed in the following order in long mode: nobody, owner, group owner, and world.

–L This option specifies that links are to be followed. If *name* actually points to a link, it is followed to the linked object.

–m Display modification time instead of creation time when listing in long format.

–M Master only. This specifies that information is to be returned from the master server of the named object. This guarantees that the most up to date information is seen at the possible expense that the master server may be busy.

–R List directories recursively. This option will reiterate the list for each subdirectory found in the process of listing each *name* .

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

EXIT CODES

nisls returns **0** on success and **1** on failure.

SEE ALSO

nisdefaults(1), **nisgrpadm**(1), **nismatch**(1), **nistbladm**(1), **nis_objects**(3N)

NAME

nismatch, nisgrep – utilities for searching NIS+ tables

SYNOPSIS

nismatch [–**AchMoPv**] *key tablename*
nismatch [–**AchMoPv**] *colname=key*. . . *tablename*
nismatch [–**AchMoPv**] *indexedname*

nisgrep [–**AchMov**] *keypat tablename*
nisgrep [–**AchMov**] *colname=keypat*. . . *tablename*

AVAILABILITY

SUNWnisu

DESCRIPTION

nismatch and **nisgrep** can be used to search NIS+ tables. The command **nisgrep** differs from the **nismatch** command in its ability to accept regular expressions *keypat* for the search criteria rather than simple text matches.

Because **nisgrep** uses a callback function, it is not constrained to searching only those columns that are specifically made searchable at the time of table creation. This makes it more flexible, but slower, than **nismatch**.

In **nismatch**, the server does the searching; wheareas in **nisgrep**, the server returns all the readable entries and then the client does the pattern-matching.

In both commands, the parameter *tablename* is the NIS+ name of the table to be searched. If only one key or key pattern is specified without the column name, then it is applied searching the first column. Specific named columns can be searched by using the *colname=key* syntax. When multiple columns are searched, only entries that match in all columns are returned. This is the equivalent of a logical join operation.

nismatch accepts an additional form of search criteria, *indexedname*, which is a NIS+ indexed name of the form:

> **[** *colname***=***value***, . . .],***tablename*

OPTIONS

–A All data. Return the data within the table and all of the data in tables in the initial table's concatenation path.

–c Print only a count of the number of entries that matched the search criteria.

–h Display a header line before the matching entries that contains the names of the table's columns

–M Master server only. Send the lookup to the master server of the named data. This guarantees that the most up to date information is seen at the possible expense that the master server may be busy.

–o Display the internal representation of the matching NIS+ object(s).

–P Follow concatenation path. Specify that the lookup should follow the concatenation path of a table if the initial search is unsuccessful.

–v Verbose. Do not suppress the output of binary data when displaying matching entries. Without this option binary data is displayed as the string ***BINARY***.

RETURN VALUES

0	Returns 0 when it successfully matches some entries.
1	Returns 1 when it successfully searches the table and no matches are found.
2	Returns 2 when an error condition occurs. An error message is also printed.

EXAMPLES

This example searches a table named **passwd** in the **org_dir** subdirectory of the **zotz.com.** domain. It returns the entry that has the username of **skippy**. In this example, all the work is done on the server.

```
example% nismatch name=skippy passwd.org_dir.zotz.com.
```

This example is similar to the one above except that it uses **nisgrep** to find all users in the table named **passwd** that are using either **ksh**(1) or **csh**(1).

```
example% nisgrep ’shell=[ck]sh’ passwd.org_dir.zotz.com.
```

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ table name is not fully qualified, each directory specified will be searched until the table is found (see **nisdefaults**(1)).

SEE ALSO

niscat(1), **nisdefaults**(1), **nisls**(1), **nistbladm**(1), **nis_objects**(3N)

DIAGNOSTICS

No memory
An attempt to allocate some memory for the search failed.

tablename **is not a table**
The object with the name *tablename* was not a table object.

Can’t compile regular expression
The regular expression in *keypat* was malformed.

column not found: *colname*
The column named *colname* does not exist in the table named *tablename*.

NAME

nismkdir – create NIS+ directories

SYNOPSIS

nismkdir [**−D** *defaults*] [**−m** *hostname* | **−s** *hostname*] *dirname*

AVAILABILITY

SUNWnisu

DESCRIPTION

The **nismkdir** command creates new NIS+ subdirectories within an existing domain. It can also be used to create replicated directories. Without options, this command will create a subdirectory with the same master and the replicas as its parent directory.

It is advisable to use **nisserver**(1M) to create an NIS+ domain which consists of the specified directory along with the **org_dir** and **group_dir** subdirectories.

The two primary aspects that are controlled when making a directory are its access rights, and its degree of replication.

dirname is the fully qualified NIS+ name of the directory that has to be created.

OPTIONS

−D *defaults*

Specify a different set of defaults to be used when creating new directories. The *defaults* string is a series of tokens separated by colons. These tokens represent the default values to be used for the generic object properties. All of the legal tokens are described below.

ttl=*time*

This token sets the default time to live for objects that are created by this command. The value *time* is specified in the format as defined by the **nischttl**(1) command. The default value is **12h** (12 hours).

owner=*ownername*

This token specifies that the NIS+ principal *ownername* should own the created object. The default for this value is the principal who is executing the command.

group=*groupname*

This token specifies that the group *groupname* should be the group owner for the object that is created. The default value is **NULL**.

access=*rights*

This token specifies the set of access rights that are to be granted for the given object. The value *rights* is specified in the format as defined by the **nischmod**(1) command. The default value is −−−−**rmcdr**−−−**r**−−−.

−m *hostname*

If the directory named by *dirname* does not exist, then a new directory that is *not* replicated is created with host *hostname* as its master server.

If the directory name by *dirname* does exist, then the host named by *hostname* is made its master server.

–**s** *hostname*	Specify that the host *hostname* will be a replica for an existing directory named *dirname.*

RETURN VALUES

0	This command returns 0 if successful and 1 otherwise.

EXAMPLES

To create a new directory **bar** under the **foo.com.** domain that shares the same master and replicas as the **foo.com.** directory one would use the command:

```
example% nismkdir bar.foo.com.
```

To create a new directory *bar.foo.com.* that is not replicated under the **foo.com.** domain one would use the command:

```
example% nismkdir –m myhost.foo.com. bar.foo.com.
```

To add a replica server of the *bar.foo.com.* directory, one would use the command:

```
example% nismkdir –s replica.foo.com. bar.foo.com.
```

ENVIRONMENT

NIS_DEFAULTS	This variable contains a defaults string that will override the NIS+ standard defaults. If the –**D** switch is used those values will then override both the **NIS_DEFAULTS** variable and the standard defaults.
NIS_PATH	If this variable is set, and the NIS+ directory name is not fully qualified, each directory specified will be searched until the directory is found (see **nisdefaults**(1)).

SEE ALSO

nis+(1), **nischmod**(1), **nisdefaults**(1), **nisls**(1), **nisrmdir**(1), **nisserver**(1M)

NOTES

A host that serves a NIS+ directory *must be* a NIS+ client in a directory above the one it is serving. The exceptions to this rule are the root NIS+ servers which are both clients and servers of the same NIS+ directory.

When the host’s default domain is different than the default domain on the client where the command is executed, the hostname supplied as an argument to the –**s** or –**m** options must be fully qualified.

NAME

nispasswd – change NIS+ password information

SYNOPSIS

nispasswd [–**ghs**] [–**D** *domainname*] [*username*]

nispasswd –**a**

nispasswd –**D** *domainname*] [–**d** [*username*]]

nispasswd [–**l**] [–**f**] [–**n** *min*] [–**x** *max*] [–**w** *warn*] [–**D** *domainname*] *username*

AVAILABILITY

SUNWnisu

DESCRIPTION

nispasswd changes a password, gecos (finger) field (–**g** option), home directory (–**h** option), or login shell (–**s** option) associated with the *username* (invoker by default) in the NIS+ passwd table.

Additionally, the command can be used to view or modify aging information associated with the user specified if the invoker has the right NIS+ privileges.

nispasswd uses secure RPC to communicate with the NIS+ server, and therefore, never sends unencrypted passwords over the communication medium.

nispasswd does not read or modify the local password information stored in the **/etc/passwd** and **/etc/shadow** files.

When used to change a password, **nispasswd** prompts non-privileged users for their old password. It then prompts for the new password twice to forestall typing mistakes. When the old password is entered, **nispasswd** checks to see if it has “aged” sufficiently. If “aging” is insufficient, **nispasswd** terminates; see **getspnam**(3C).

The old password is used to decrypt the username’s secret key. If the password does not decrypt the secret key, **nispasswd** prompts for the old secure-RPC password. It uses this password to decrypt the secret key. If this fails, it gives the user one more chance. The old password is also used to ensure that the new password differs from the old by at least three characters. Assuming aging is sufficient, a check is made to ensure that the new password meets construction requirements described below. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical, the cycle of prompting for the new password is repeated twice. The new password is used to re-encrypt the user’s secret key. Hence, it also becomes their secure-RPC password.

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant.
- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" refers to all upper or lower case letters.
- Each password must differ from the user’s login *username* and any reverse or circular shift of that login *username*. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

- New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

Network administrators, who own the NIS+ password table, may change any password attributes if they establish their credentials (see **keylogin**(1)) before invoking **nispasswd**. Hence, **nispasswd** does not prompt these privileged-users for the old password and they are not forced to comply with password aging and password construction requirements.

Any user may use the −**d** option to display password attributes for his or her own login name. The format of the display will be:

username status mm/dd/yy min max warn

or, if password aging information is not present,

username status

where

username	The login ID of the user.
status	The password status of *username*: "PS" stands for password exists or locked, "LK" stands for locked, and "NP" stands for no password.
mm/dd/yy	The date password was last changed for *username.* (Note that all password aging dates are determined using Greenwich Mean Time and, therefore, may differ by as much as a day in other time zones.)
min	The minimum number of days required between password changes for *username.*
max	The maximum number of days the password is valid for *username.*
warn	The number of days relative to *max* before the password expires that the *username* will be warned.

OPTIONS

−**g**	Change the gecos (finger) information.
−**h**	Change the home directory.
−**s**	Change the login shell. By default, only the NIS+ administrator can change the login shell. User will be prompted for the new login shell.
−**a**	Show the password attributes for all entries. This will show only the entries in the NIS+ passwd table in the local domain that the invoker is authorized to "read".
−**d** [*username*]	Display password attributes for the caller or the user specified if the invoker has the right privileges.
−**l**	Locks the password entry for *username.* Subsequently, **login**(1) would disallow logins with this NIS+ password entry.
−**f**	Force the user to change password at the next login by expiring the password for *username.*

–n *min*	Set minimum field for *username*. The *min* field contains the minimum number of days between password changes for *username*. If *min* is greater than *max*, the user may not change the password. Always use this option with the **–x** option, unless *max* is set to -1 (aging turned off). In that case, *min* need not be set.
–x *max*	Set maximum field for *username*. The *max* field contains the number of days that the password is valid for *username*. The aging for *username* will be turned off immediately if *max* is set to -1. If it is set to 0, then the user is forced to change the password at the next login session and aging is turned off.
–w *warn*	Set *warn* field for *username*. The *warn* field contains the number of days before the password expires that the user will be warned whenever he or she attempts to login.
–D *domainname*	Consult the **passwd.org_dir** table in *domainname*. If this option is not specified, the default domainname returned by **nis_local_directory( )** will be used. This domainname is the same as that returned by **domainname**(1M).

SEE ALSO

keylogin(1), **login**(1), **nis+**(1), **nistbladm**(1), **passwd**(1), **domainname**(1M), **getspnam**(3C), **getpwnam**(3C), **nsswitch.conf**(4), **passwd**(4), **shadow**(4)

DIAGNOSTICS

The **nispasswd** command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. NIS+ passwd table unchanged.
4	NIS+ passwd table missing.
5	NIS+ is busy. Try again later.
6	Invalid argument to option.
7	Aging is disabled.

NOTES

The login program, file access display programs (for example, '**ls –l**') and network programs that require user passwords (for example, **rlogin**(1), **ftp**(1), etc.) use the standard **getpwnam**(3C) and **getspnam**(3C) interfaces to get password information. These programs will get the NIS+ password information, that is modified by **nispasswd**, only if the **passwd:** entry in the **/etc/nsswitch.conf** file includes to **nisplus**. See **nsswitch.conf**(4) for more details.

NAME

nisrm – remove NIS+ objects from the namespace

SYNOPSIS

nisrm [**–if**] *name* . . .

AVAILABILITY

SUNWnisu

DESCRIPTION

The **nisrm** command removes NIS+ objects named *name* from the NIS+ namespace.

This command will fail if the NIS+ master server is not running.

OPTIONS

–i Interactive mode. Like the system **rm**(1) command the **nisrm** command will ask for confirmation prior to removing an object. If the name specified by *name* is a non-fully qualified name this option is forced on. This prevents the removal of unexpected objects.

–f Force. The removal is attempted, and if it fails for permission reasons, a **nischmod**(1) is attempted and the removal retried. If the command fails, it fails silently.

EXAMPLES

Remove the objects *foo*, *bar*, and *baz* from the namespace.

```
example% nisrm foo bar baz
```

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

EXIT CODES

nisrm returns 0 on success and 1 on failure.

SEE ALSO

nis+(1), **nischmod**(1), **nisdefaults**(1), **nisrmdir**(1), **nistbladm**(1), **rm**(1)

NOTES

This command will not remove directories (see **nisrmdir**(1)) nor will it remove non-empty tables (see **nistbladm**(1)).

NAME

nisrmdir – remove NIS+ directories

SYNOPSIS

nisrmdir [**–if**] [**–s** *hostname*] *dirname*

AVAILABILITY

SUNWnisu

DESCRIPTION

nisrmdir deletes existing NIS+ subdirectories. It can remove a directory outright, or simply remove replicas from serving a directory.

This command modifies the object that describes the directory *dirname*, and then notifies each replica to remove the directory named *dirname.* If the notification of any of the affected replicas fails, the directory object is returned to its original state unless the **–f** option is present.

This command will fail if the NIS+ master server is not running.

OPTIONS

–i Interactive mode. Like the system **rm**(1) command the **nisrmdir** command will ask for confirmation prior to removing a directory. If the name specified by *dirname* is a non-fully qualified name this option is forced on. This prevents the removal of unexpected directories.

–f Force the command to succeed even though it may not be able to contact the affected replicas. This option should be used when a replica is known to be down and will not be able to respond to the removal notification. When the replica is finally rebooted it will read the updated directory object, note that it is no longer a replica for that directory, and stop responding to lookups on that directory. Cleanup of the files that held the now removed directory can be accomplished manually by removing the appropriate files in the */var/nis* directory (see **nisfiles**(4) for more information).

–s *hostname* Specify that the host *hostname* should be removed as a replica for the directory named *dirname*. If this option is not present *all* replicas and the master server for a directory are removed and the directory is removed from the namespace.

RETURN VALUES

This command returns 0 if it is successful, and 1 otherwise.

EXAMPLES

To remove a directory **bar** under the **foo.com.** domain, one would use the command:

```
example% nisrmdir bar.foo.com.
```

To remove a replica that is serving directory **bar.foo.com.** one would use the command:

```
example% nisrmdir –s replica.foo.com. bar.foo.com.
```

To force the removal of directory **bar.foo.com.** from the namespace, one would use the command:

```
example% nisrmdir –f bar.foo.com.
```

ENVIRONMENT

NIS_PATH	If this variable is set, and the NIS+ directory name is not fully qualified, each directory specified will be searched until the directory is found (see **nisdefaults**(1)).

SEE ALSO

nis+(1), **nisdefaults**(1), **nisrm**(1), **nisfiles**(4)

NAME

nistbladm – NIS+ table administration command

SYNOPSIS

nistbladm –a | **–A** [**–D** *defaults*] *colname=value . . . tablename*

nistbladm –a | **–A** [**–D** *defaults*] *indexedname*

nistbladm –c [**–D** *defaults*] [**–p** *path*] [**–s** *sep*] *type colname*=[*flags*][,*access*] . . . *tablename*

nistbladm –d *tablename*

nistbladm –m *colname=value . . . indexedname*

nistbladm –r | **–R** [*colname=value . . .*] *tablename*

nistbladm –r | **–R** *indexedname*

nistbladm –u [**–p** *path*] [**–s** *sep*] [**–t** *type*] [*colname*=*access . . .*] *tablename*

AVAILABILITY

SUNWnisu

DESCRIPTION

The **nistbladm** command is used to administer NIS+ tables. There are five primary operations that it performs: creating and deleting tables, adding entries to, modifying entries within, and removing entries from tables.

Though NIS+ does not place restrictions on the size of tables or entries, the size of data has an impact on the performance and the disk space requirements of the NIS+ server. NIS+ is not designed to store huge pieces of data, such as files; instead pointer to files should be stored in NIS+.

NIS+ design is optimized to support 10,000 objects with a total size of 10M bytes. If the requirements exceed the above, it is suggested that the domain hierarchy be created, or the data stored in the tables be pointers to the actual data, instead of the data itself.

When creating tables, a table type, *type*, and a list of column definitions must be provided.

type is a string that is stored in the table and later used by the service to verify that entries being added to it are of the correct type.

Syntax for column definitions is:

colname=[*flags*][,*access*]

flags is a combination of:

S	Searchable. Specifies that searches can be done on the column's values (see **nismatch**(1)).
I	Case-insensitive (only makes sense in combination with **S**). Specifies that searches should ignore case.
C	Crypt. Specifies that the column's values should be encrypted.
B	Binary data (does not make sense in combination with **S**). If not set, the column's values are expected to be null terminated ASCII strings.

X XDR encoded data (only makes sense in combination with **B**).

access is specified in the format as defined by the **nischmod**(1) command.

When manipulating entries, this command takes two forms of entry name. The first uses a series of space separated *colname*=*value* pairs that specify column values in the entry. The second is a NIS+ indexed name, *indexedname*, of the form:

[*colname*=*value*, . . . **],***tablename*

OPTIONS

−a | A Add entries to a NIS+ table. The difference between the lowercase ‘a’ and the uppercase ‘A’ is in the treatment of preexisting entries. The entry’s contents are specified by the *column*=*value* pairs on the command line. Note: Values for *all* columns must be specified when adding entries to a table.

Normally, NIS+ reports an error if an attempt is made to add an entry to a table that would overwrite an entry that already exists. This prevents multiple parties from adding duplicate entries and having one of them get overwritten. If you wish to force the add, the uppercase ‘A’ specifies that the entry is to be added, even if it already exists. This is analogous to a modify operation on the entry.

−c Create a table named *tablename* in the namespace. The table that is created must have at least one column and at least one column must be searchable.

−d *tablename*

Destroy the table named *tablename*. The table that is being destroyed must be empty. The table’s contents can be deleted with the **−R** option below.

−m Modify an entry in the table that is specified by *indexedname*. Note: Since it is possible to modify the value in a column that would change the indexed name for an entry both the column value pair and the indexed name are required. It uses the indexed name to look up the entry, modify it, and write it back with the new value. The indexed name must uniquely identify a single entry.

−r |R Remove entries from a table. The entry is specified by either a series of *column*=*value* pairs on the command line, or an indexed name that is specified as *entryname*. The difference between the interpretation of the lowercase ‘r’ versus the uppercase ‘R’ is in the treatment of non-unique entry specifications. Normally the NIS+ server will disallow an attempt to remove an entry when the search criterion specified for that entry resolves to more than one entry in the table. However, it is sometimes desirable to remove more than one entry, as when you are attempting to remove all of the entries from a table. In this case, using the uppercase ‘R’ will force the NIS+ server to remove all entries matching the passed search criterion. If that criterion is null and no column values specified, then all entries in the table will be removed.

−u Update attributes of a table. This allows the concatenation path (**−p**), separation character (specified with the (**−s**), column access rights, and table type string (**−t**) of a table to be changed. Neither the number of columns, nor the columns that are searchable may be changed.

−D *defaults*

When creating objects, this option specifies a different set of defaults to be used during this operation. The *defaults* string is a series of tokens separated by colons. These tokens represent the default values to be used for the generic object properties. All of the legal tokens are described below.

ttl=*time* This token sets the default time to live for objects that are created by this command. The value *time* is specified in the format as defined by the **nischttl**(1) command. The default value is 12 hours.

owner=*ownername*
This token specifies that the NIS+ principal *ownername* should own the created object. Normally this value is the same as the principal who is executing the command.

group=*groupname*
This token specifies that the group *groupname* should be the group owner for the object that is created. The default value is NULL.

access=*rights*
This token specifies the set of access rights that are to be granted for the given object. The value *rights* is specified in the format as defined by the **nischmod**(1) command. The default value is **−−−−rmcdr−−−r−−−**.

−p *path* When creating or updating a table, this option specifies the table's search path. When a **nis_list( )** function is invoked, the user can specify the flag FOLLOW_PATH to tell the client library to continue searching tables in the table's path if the search criteria used does not yield any entries. The path consists of an ordered list of table names, separated by colons. The names in the path must be fully qualified.

−s *sep* When creating or updating a table, this option specifies the table's separator character. The separator character is used by **niscat**(1) when displaying tables on the standard output. Its purpose is to separate column data when the table is in ASCII form. The default value is a space.

−t *type* When updating a table, this option specifies the table's type string.

RETURN VALUES

This command returns **0** on success and **1** on failure.

EXAMPLES

This example creates a table named **hobbies** in the directory **foo.com.** of the type **hobby_tbl** with two searchable columns, **name** and **hobby**.

```
example% nistbladm −c hobby_tbl name=S,a+r,o+m hobby=S,a+r hobbies.foo.com.
```

The column **name** has read access for all (that is, **owner**, **group**, and **world**) and modify access for only the owner. The column **hobby** is readable by all, but not modifiable by anyone.

In this example, if the access rights had not been specified, the tables access rights would have come from either the standard defaults or the **NIS_DEFAULTS** variable (see below).

To add entries to this table:

```
example% nistbladm –a name=bob hobby=skiing hobbies.foo.com.
example% nistbladm –a name=sue hobby=skiing hobbies.foo.com.
example% nistbladm –a name=ted hobby=swimming hobbies.foo.com.
```

To add the concatenation path:

```
example% nistbladm –u –p hobbies.bar.com.:hobbies.baz.com. hobbies
```

To delete the skiers from our list:

```
example% nistbladm –R hobby=skiing hobbies.foo.com.
```

Note: The use of the **–r** option would fail because there are two entries with the value of **skiing**.

To create a table with a column that is named with no flags set, you supply only the name and the equals (=) sign as follows.

```
example% nistbladm –c notes_tbl name=S,a+r,o+m note=  notes.foo.com.
```

This example created a table, named *notes.foo.com.*, of type *notes_tbl* with two columns **name** and **note**. The **note** column is not searchable.

When entering data for columns in the form of a *value* string, it is essential that terminal characters be protected by single or double quotes. These are the characters equals (=), comma (,), left bracket ([), right bracket (]), and space (). These characters are parsed by NIS+ within an indexed name. These characters are protected by enclosing the entire value in double quote (") characters as follows.

```
example% nistbladm –a fullname="Joe User" nickname=Joe nicknames
```

If there is any doubt about how the string will be parsed, it is better to enclose it in quotes.

ENVIRONMENT

NIS_DEFAULTS This variable contains a defaults string that will be override the NIS+ standard defaults. If the **–D** switch is used those values will then override both the **NIS_DEFAULTS** variable and the standard defaults.

NIS_PATH If this variable is set, and the NIS+ table name is not fully qualified, each directory specified will be searched until the table is found (see **nisdefaults**(1)).

SEE ALSO

nis+(1), **niscat**(1), **nischmod**(1), **nischown**(1), **nisdefaults**(1), **nismatch**(1), **nissetup**(1M)

WARNINGS

To modify one of the entries, say, for example, from “bob” to “robert”:

```
example% nistbladm -m name=robert [name=bob],hobbies
```

Note that “**[name=bob],hobbies**” is an indexed name, and that the characters ‘[’ (open bracket) and ‘]’ (close bracket) are interpreted by the shell. When typing entry names in the form of NIS+ indexed names, the name must be protected by using single quotes.

It is possible to specify a set of defaults such that you cannot read or modify the table object later.

NAME

nistest – return the state of the NIS+ namespace using a conditional expression

SYNOPSIS

nistest [–**ALMP**] [–**a** *rights* | –**t** *type*] *object*

nistest [–**ALMP**] [–**a** *rights*] *indexedname*

AVAILABILITY

SUNWnisu

DESCRIPTION

nistest provides a way for shell scripts and other programs to test for the existence, type, and access rights of objects and entries. Entries are named using indexed names (see **nismatch**(1)).

OPTIONS

–**A**	All data. This option specifies that the data within the table and all of the data in tables in the initial table's concatenation path be returned. This option is only valid when using indexed names or following links.
–**L**	Follow links. If the object named by *object* or the tablename component of *indexedname* names a LINK type object, the link is followed when this switch is present.
–**M**	Master server only. This option specifies that the lookup should be sent to the master server of the named data. This guarantees that the most up to date information is seen at the possible expense that the master server may be busy.
–**P**	Follow concatenation path. This option specifies that the lookup should follow the concatenation path of a table if the initial search is unsuccessful. This option is only valid when using indexed names or following links.
–**a** *rights*	This option is used to verify that the current process has the desired or required access rights on the named object or entries. The access rights are specified in the same way as the **nischmod**(1) command.
–**t** *type*	This option tests the type of *object*. The value of *type* can be one of the following:

G	Return true if the object is a group object.
D	Return true if the object is a directory object.
T	Return true if the object is a table object.
L	Return true if the object is a link object.
P	Return true if the object is a private object.

RETURN VALUES

0	Success.
1	Failure due to object not present, not of specified type and/or no such access.
2	Failure due to illegal usage.

EXAMPLES

When testing for access rights, **nistest** returns success (0) if the specified rights are granted to the current user. Thus testing for access rights

```
example% nistest –a w=mr skippy.domain
```

Tests that all authenticated NIS+ clients have read and modify access to the object named *skippy.domain*.

Testing for access on a particular entry in a table can be accomplished using the indexed name syntax. The following example tests to see if an entry in the password table can be modified.

```
example% nistest –a o=m ’[uid=99],passwd.org_dir’
```

ENVIRONMENT

NIS_PATH If this variable is set, and the NIS+ name is not fully qualified, each directory specified will be searched until the object is found (see **nisdefaults**(1)).

SEE ALSO

nis+(1), **nischmod**(1), **nisdefaults**(1)

NAME

nl – line numbering filter

SYNOPSIS

nl [–**b***type*] [–**f***type*] [–**h***type*] [–**v***start#*] [–**i***incr*] [–**p**] [–**l***num*] [–**s***sep*] [–**w***width*] [–**n***format*] [–**d***delim*] [*filename*]

AVAILABILITY

SUNWesu

DESCRIPTION

nl reads lines from the named *filename*, or the standard input if no *filename* is named, and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer. For example, –**bt** (the default) numbers non-blank lines in the body section and does not number any lines in the header and footer sections.

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, **nl** assumes the text being read is in a single logical page body.

OPTIONS

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

–**b***type* Specifies which logical page body lines are to be numbered. Recognized *type*s and their meanings are:

a	number all lines
t	number lines with printable text only
n	no line numbering
p*exp*	number only lines that contain the regular expression specified in *exp* (see **ed**(1))

Default *type* for logical page body is **t** (text lines numbered).

–**f***type* Same as –**b***type* except for footer. Default *type* for logical page footer is **n** (no lines numbered).

–**h***type* Same as –**b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

–**v***start#*	*start#* is the initial value used to number logical page lines. Default *start#* is **1**.
–**i***incr*	*incr* is the increment value used to number logical page lines. Default *incr* is **1**.
–**p**	Do not restart numbering at logical page delimiters.
–**l***num*	*num* is the number of blank lines to be considered as one. For example, –**l2** results in only the second adjacent blank being numbered (if the appropriate –**ha**, –**ba**, and / or –**fa** option is set). Default *num* is **1**.
–**s***sep*	*sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
–**w***width*	*width* is the number of characters to be used for the line number. Default *width* is **6**.
–**n***format*	*format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).
–**d***delim*	The two delimiter characters specifying the start of a logical page section may be changed from the default characters (\ :) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the –**d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLES

The command:

```
example% nl –v10 –i10 –d!+ filename1
```

will cause the first line of the page body to be numbered **10**, the second line of the page body to be numbered **20**, the third **30**, and so forth. The logical page delimiters are !+.

SEE ALSO

ed(1), **pr**(1)

NAME

nm – print name list of an object file

SYNOPSIS

nm [**–CefhlnoprRsTuvVx**] *filename*. . .

DESCRIPTION

The **nm** command displays the symbol table of each ELF object file that is specified by *filename*(s). For each symbol, the following information will be printed:

Index	The index of the symbol. (The index appears in brackets.)
Value	The value of the symbol is one of the following: a section offset for defined symbols in a relocatable file; alignment constraints for symbols whose section index is **SHN_COMMON**; a virtual address in executable and dynamic library files.
Size	The size in bytes of the associated object.
Type	A symbol is of one of the following types: **NOTYPE** (no type was specified), **OBJECT** (a data object such as an array or variable), **FUNC** (a function or other executable code), **SECTION** (a section symbol), or **FILE** (name of the source file).
Bind	The symbol's binding attributes. **LOCAL** symbols have a scope limited to the object file containing their definition; **GLOBAL** symbols are visible to all object files being combined; and **WEAK** symbols are essentially global symbols with a lower precedence than **GLOBAL**.
Other	A field reserved for future use, currently containing 0.
Shndx	Except for three special values, this is the section header table index in relation to which the symbol is defined. The following special values exist: **ABS** indicates the symbol's value will not change through relocation; **COMMON** indicates an unallocated block and the value provides alignment constraints; and **UNDEF** indicates an undefined symbol.
Name	The name of the symbol.

OPTIONS

The output of **nm** may be controlled using the following options:

–C	Demangle C++ symbol names before printing them out.
–e	See **NOTES** below.
–f	See NOTES below.
–h	Do not display the output heading data.
–l	Distinguish between **WEAK** and **GLOBAL** symbols by appending a * to the key letter for **WEAK** symbols.
–n	Sort external symbols by name before they are printed.
–o	Print the value and size of a symbol in octal instead of decimal.
–p	Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **N** (symbol

has no type), **D** (data object symbol), **B** (bss symbol), **T** (text symbol), **S** (section symbol), or **F** (file symbol). If the symbol's binding attribute is **LOCAL**, the key letter is lower case; if the symbol's binding attribute is **WEAK ,** the key letter is upper case; if the **–l** modifier is specified, the upper case key letter is followed by a *; if the symbol's binding attribute is **GLOBAL ,** the key letter is upper case.

–r Prepend the name of the object file or archive to each output line.

–R Print the archive name (if present), followed by the object file and symbol name. If the **–r** option is also specified, this option is ignored.

–s Print section name instead of section index.

–T See NOTES below.

–u Print undefined symbols only.

–v Sort external symbols by value before they are printed.

–V Print the version of the **nm** command executing on the standard error output.

–x Print the value and size of a symbol in hexadecimal instead of decimal.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. When conflicting options are specified (such as **–v** and **–n**; and **–o** and **–x**) the first is taken and the second ignored with a warning message to the user. (See **–R** for exception.)

SEE ALSO

as(1), **dump**(1), **ld**(1), **a.out**(4), **ar**(4)

NOTES

The following options are obsolete because of changes to the object file format and will be deleted in a future release.

–e Print only external and static symbols. The symbol table now contains only static and external symbols. Automatic symbols no longer appear in the symbol table. They do appear in the debugging information produced by **cc –g**, which may be examined using **dump**(1).

–f Produce full output. Redundant symbols (such as .text, .data, and so forth). which existed previously do not exist and producing full output will be identical to the default output.

–T By default, **nm** prints the entire name of the symbols listed. Since symbol names have been moved to the last column, the problem of overflow is removed and it is no longer necessary to truncate the symbol name.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

/usr/bin/nohup *command* [*arguments*]

AVAILABILITY

SUNWcsu

DESCRIPTION

Processes run by **nohup** are immune to HUP (hangup) and QUIT (quit) signals; **nohup** does not arrange to make them immune to a TERM (terminate) signal, so unless they arrange to be immune to a TERM signal, or the shell makes them immune to a TERM signal, they will receive that signal. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**. If the standard error is a terminal, it is redirected to the standard output, otherwise it is not redirected. The priority of the process run by **nohup** is not altered.

EXAMPLES

It is frequently desirable to apply **nohup** to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
$ nohup sh filename
```

and the **nohup** applies to everything in *filename*. If the shell procedure *filename* is to be executed often, then the need to type **sh** can be eliminated by giving *filename* execute permission. Add an ampersand and the contents of *filename* are run in the background with interrupts also ignored (see **sh**(1)):

```
$ nohup filename &
```

An example of what the contents of *filename* could be is:

```
sort ofilename > nfilename
```

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **nohup** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **nohup** behaves.

LC_CTYPE

Determines how **nohup** handles characters. When **LC_CTYPE** is set to a valid value, **nohup** can display and handle text and filenames containing valid characters for that locale. **nohup** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **nohup** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

chmod(1), **csh**(1), **nice**(1), **sh**(1), **shell_builtins**(1), **signal**(3C), **environ**(5)

NOTES

If you use **csh**(1), then commands executed with '**&**' are automatically immune to HUP signals while in the background.

There is a C shell built-in command **nohup** that provides immunity from terminate, but does not redirect output to **nohup.out**.

nohup does not recognize command sequences. In the case of the following command

```
$ nohup command1; command2
```

nohup applies only to **command1**. The command

```
$ nohup (command1; command2)
```

is syntactically incorrect.

NAME

nroff – format documents for display or line-printer

SYNOPSIS

nroff [–**ehiq**] [–**m***name*] [–**n***N*] [–**o***pagelist*] [–**r***aN*] [–**s***N*] [–**T***name*]

AVAILABILITY

SUNWdoc

DESCRIPTION

nroff formats text in the named *files* for typewriter-like devices. See also **troff**(1).

If no *file* argument is present, **nroff** reads the standard input. An argument consisting of a '–' is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear *before* the files.

–e Produce equally-spaced words in adjusted lines, using full terminal resolution.

–h Use output TAB characters during horizontal spacing to speed output and reduce output character count. TAB settings are assumed to be every 8 nominal character widths.

–i Read the standard input after the input files are exhausted.

–q Invoke the simultaneous input-output mode of the **rd** request.

–m*name*
Prepend the macro file **/usr/share/lib/tmac/tmac.***name* to the input files.

–n*N* Number first generated page *N*.

–o*pagelist*
Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end.

–r*aN* Set register *a* (one-character) to *N*.

–s*N* Stop every *N* pages. **nroff** will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a NEWLINE.

–T*name*
Prepare output for a device of the specified *name*. Known *name*s are:

37	Teletype Corporation Model 37 terminal — this is the default.
lp | **tn300**	GE Any line printer or terminal without half-line capability.
300	DASI-300.
300-12	DASI-300 — 12-pitch.
300S	DASI-300S.
300S-12	DASI-300S.
382	DASI-382 (fancy DTC 382).
450	DASI-450 (Diablo Hyterm).
450-12	DASI-450 (Diablo Hyterm) — 12-pitch.
832	AJ 832.

EXAMPLE

The following command:

```
example% nroff –s4 –me users.guide
```

formats **users.guide** using the –**me** macro package, and stopping every 4 pages.

FILES

/var/tmp/trtmp∗	temporary file
/usr/share/lib/tmac/tmac.∗	standard macro files
/usr/share/lib/nterm/∗	terminal driving tables for **nroff**
/usr/share/lib/nterm/README	index to terminal description files

SEE ALSO

checknr(1), **col**(1), **eqn**(1), **man**(1), **tbl**(1), **troff**(1), **term**(5), **me**(5), **ms**(5)

NOTES

nroff is not 8-bit clean because making **nroff** 8-bit clean would require rewriting the **nroff** internals and filters. Also, some **nroff** syntax is based on ASCII only and does not lend itself to 8-bit character sequences.

NAME

od – octal dump

SYNOPSIS

od [–**bcCDdFfOoSsvXx**] [*filename*] [[+] *offset* [**.**] [**b**]]

DESCRIPTION

od displays *filename* in one or more formats, as selected by the first argument. If the first argument is missing, –**o** is default. If no *filename* is specified, the standard input is used. For the purposes of this description, "word" refers to a 16-bit unit, independent of the word size of the machine; "long word" refers to a 32-bit unit, and "double long word" refers to a 64-bit unit.

OPTIONS

–**b**	Interpret bytes in octal.
–**c**	Display single-byte characters. Certain non-graphic characters appear as C-language escapes: null=**\0**, backspace=**\b**, form-feed=**\f**, new-line=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers. For example:

```
echo "hello world" | od -c
0000000   h   e   l   l   o       w   o   r   l   d  \n
0000014
```

–**C**	Interpret bytes as single-byte or multibyte characters according to the current setting of the LC_CTYPE locale category. Printable multibyte characters are written in the area corresponding to the first byte of the character; two character sequence ** are written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. Non graphic characters appear as same as they would using the –**c** option.
–**D**	Interpret long words in unsigned decimal.
–**d**	Interpret words in unsigned decimal.
–**F**	Interpret double long words in extended precision.
–**f**	Interpret long words in floating point.
–**O**	Interpret long words in unsigned octal.
–**o**	Interpret words in octal.
–**S**	Interpret long words in signed decimal.
–**s**	Interpret words in signed decimal.
–**v**	Show all data (verbose).
–**X**	Interpret long words in hex.
–**x**	Interpret words in hex.

offset specifies an offset from the beginning of *filename* where the display will begin. *offset* is normally interpreted as octal bytes. If **.** is appended, *offset* is interpreted in decimal. If **b** is appended, *offset* is interpreted in blocks of 512 bytes. If *filename* is omitted, *offset* must be preceded by **+**.

The display continues until an end-of-file is reached.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **od** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **od** behaves.

LC_CTYPE

Determines how **od** handles characters. When **LC_CTYPE** is set to a valid value, **od** can display and handle text and filenames containing valid characters for that locale. **od** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **od** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

environ(5)

NAME

on – execute a command on a remote system, but with the local environment

SYNOPSIS

on [**–i**] [**–d**] [**–n**] *host command* [*argument*] . . .

AVAILABILITY

SUNWcsu

DESCRIPTION

The **on** program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

The standard input is connected to the standard input of the remote command, and the standard output and the standard error from the remote command are sent to the corresponding files for the **on** command.

OPTIONS

–i	Interactive mode. Use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.
–d	Debug mode. Print out some messages as work is being done.
–n	No Input. This option causes the remote program to get EOF when it reads from the standard input, instead of passing the standard input from the standard input of the **on** program. For example, **–n** is necessary when running commands in the background with job control.

SEE ALSO

chkey(1)

DIAGNOSTICS

unknown host	Host name not found.
cannot connect to server	Host down or not running the server.
can't find	Problem finding the working directory.
can't locate mount point	Problem finding current file system.
RPC: Authentication error	The server requires des authentication and you do not have a secret key registered with keyserv. Perhaps you logged in without a password. Try to keylogin. If that fails try to set your publickey with chkey.

Other error messages may be passed back from the server.

BUGS

When the working directory is remote mounted over NFS, a **^Z** hangs the window.

Root cannot use **on**.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [–**f**] *name* . . .

pcat *name* . . .

unpack *name* . . .

AVAILABILITY

SUNWesu

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name***.z** with the same access modes, access and modified dates, and owner as those of *name*.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called *name***.z** already exists;
the **.z** file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

pcat does for packed files what **cat**(1) does for ordinary files, except that **pcat** cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named **name.z** use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say **nnn**, of a packed file named **name.z** (without destroying **name.z**) use the command:

```
pcat name >nnn
```

pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the **.z**) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of **pack**.

unpack expands files created by **pack**. For each file *name* specified in the command, a search is made for a file called *name***.z** (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in **pcat**, as well as for the following:

a file with the ‘‘unpacked’’ name already exists;
if the unpacked file cannot be created.

OPTIONS

–f Forces packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If **pack** is successful, *name* will be removed. Packed files can be restored to their original form using **unpack** or **pcat**.

– **pack** uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **pack** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **pack** behaves.

LC_CTYPE

Determines how **pack** handles characters. When **LC_CTYPE** is set to a valid value, **pack** can display and handle text and filenames containing valid characters for that locale. **pack** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **pack** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

cat(1), **compress**(1), **environ**(5)

NAME

pagesize – display the size of a page of memory

SYNOPSIS

/usr/ucb/pagesize

AVAILABILITY

SUNWscpu

DESCRIPTION

pagesize prints the size of a page of memory in bytes, as returned by **getpagesize**. This program is useful in constructing portable shell scripts.

SEE ALSO

getpagesize(3B)

NAME

passwd – change login password and password attributes

SYNOPSIS

passwd [*name*]

passwd [**–d** | **–l**] [**–f**] [**–n** *min*] [**–w** *warn*] [**–x** *max*] *name*

passwd **–s** [**–a**]

passwd **–s** [*name*]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **passwd** command changes the password or lists password attributes associated with the user's login *name*. Additionally, privileged-users may use **passwd** to install or change passwords and attributes associated with any login *name*.

When used to change a password, **passwd** prompts ordinary users for their old password, if any. It then prompts for the new password twice. When the old password is entered, **passwd** checks to see if it has "aged" sufficiently. If "aging" is insufficient, **passwd** terminates; see **pwconv**(1M) and **shadow**(4) for additional information. The **pwconv** command creates and updates **/etc/shadow** with information from **/etc/passwd**. **pwconv** relies on a special value of 'x' in the password field of **/etc/passwd**. This value of 'x' indicates that the password for the user is already in **/etc/shadow** and should not be modified.

Assuming aging is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant. PASSLENGTH is found in **/etc/default/passwd** and is set to 6.
- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" refers to all upper or lower case letters.
- Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.
- New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

Super-users (for instance, real and effective uid equal to zero, see **id**(1M) and **su**(1M)) may change any password; hence, **passwd** does not prompt privileged-users for the old password. Privileged-users are not forced to comply with password aging and password construction requirements. A privileged-user can create a null password by entering a carriage return in response to the prompt for a new password. (This differs from **passwd –d** because the "password" prompt will still be displayed.)

Any user may use the **–s** option to show password attributes for his or her own login *name*.

The format of the display will be:

name status mm/dd/yy min max warn

or, if password aging information is not present,

name status

where

name	The login ID of the user.
status	The password status of *name*: "PS" stands for passworded or locked, "LK" stands for locked, and "NP" stands for no password.
mm/dd/yy	The date password was last changed for *name*. (Note: All password aging dates are determined using Greenwich Mean Time and, therefore, may differ by as much as a day in other time zones.)
min	The minimum number of days required between password changes for *name*. MINWEEKS is found in **/etc/default/passwd** and is set to NULL.
max	The maximum number of days the password is valid for *name*. MAXWEEKS is found in **/etc/default/passwd** and is set to NULL.
warn	The number of days relative to *max* before the password expires that the *name* will be warned.

OPTIONS

Only a privileged-user can use the following options:

–a	Show password attributes for all entries. Use only with **–s** option; *name* must not be provided.
–d	Deletes password for *name*. The login *name* will not be prompted for password.
–f	Force the user to change password at the next login by expiring the password for *name*.
–l	Locks password entry for *name*.
–s	Show password attributes for the login *name*.

−n *min* Set minimum field for *name*. The *min* field contains the minimum number of days between password changes for *name*. If *min* is greater than *max*, the user may not change the password. Always use this option with the **−x** option, unless *max* is set to −1 (aging turned off). In that case, *min* need not be set.

−w *warn* Set warn field for *name*. The *warn* field contains the number of days before the password expires that the user will be warned.

−x *max* Set maximum field for *name*. The *max* field contains the number of days that the password is valid for *name*. The aging for *name* will be turned off immediately if *max* is set to -1. If it is set to 0, then the user is forced to change the password at the next login session and aging is turned off.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **passwd** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **passwd** behaves.

LC_CTYPE Determines how **passwd** handles characters. When **LC_CTYPE** is set to a valid value, **passwd** can display and handle text and filenames containing valid characters for that locale. **passwd** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **passwd** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/etc/oshadow
/etc/passwd
/etc/shadow
/etc/default/passwd Default values can be set for the following flags in **/etc/default/passwd**. For example: MAXWEEKS=26

MAXWEEKS	Maximum time period that password is valid.
MINWEEKS	Minimum time period before the password can be changed.
PASSLENGTH	Minimum length of password, in characters.
WARNWEEKS	Time period until warning of date of password’s ensuing expiration.

SEE ALSO

finger(1), **login**(1), **nispasswd**(1), **yppasswd**(1), **domainname**(1M), **eeprom**(1M), **id**(1M), **passmgmt**(1M), **pwconv**(1M), **su**(1M), **useradd**(1M), **userdel**(1M), **usermod**(1M), **crypt**(3C), **getpwnam**(3C), **getspnam**(3C), **loginlog**(4), **passwd**(4), **shadow**(4), **environ**(5)

DIAGNOSTICS

The **passwd** command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. Password file unchanged.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.
6	Invalid argument to option.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste *filename1 filename2*. . .
paste –**d** *list filename1 filename2*. . .
paste –**s** [–**d** *list*] *filename1 filename2*. . .

AVAILABILITY

SUNWesu

DESCRIPTION

In the first two forms, **paste** concatenates corresponding lines of the given input files *filename1*, *filename2*, and so forth. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of **cat**(1) which concatenates vertically, that is, one file after the other. In the last form above, **paste** replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

OPTIONS

–**d** *list*	Without this option, the new-line characters of each but the last file (or last line in case of the –**s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below). One or more characters immediately following –**d** replace the default *tab* as the line concatenation character. The list is used circularly, that is, when exhausted, it is reused. In parallel merging (that is, no –**s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), \\ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (for instance, to get one backslash, use *"" –d "\\\\"*).
–**s**	Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with –**d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
–	May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

The first example will list a directory in one column.

```
ls | paste –d" " –
```

The second example will list a directory in four columns.

```
ls | paste – – – –
```

The next example will combine pairs of lines into lines.

```
paste −s −d"\t\n" file
```

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **paste** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **paste** behaves.

LC_CTYPE
Determines how **paste** handles characters. When **LC_CTYPE** is set to a valid value, **paste** can display and handle text and filenames containing valid characters for that locale. **paste** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **paste** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

cut(1), **grep**(1), **pr**(1), **environ**(5)

DIAGNOSTICS

"**line too long**"	Output lines are restricted to 511 characters.
"**too many files**"	Except for −**s** option, no more than 12 input files may be specified.

NAME

pathconv – search FMLI criteria for filename

SYNOPSIS

pathconv [–**f**] [–**v** *alias*]
pathconv [–**t**] [–**l**] [–**n***num*] [–**v** *string*]

DESCRIPTION

The **pathconv** function converts an alias to its pathname. By default, it takes the alias as a string from the standard input.

OPTIONS

–**f** If –**f** is specified, the full path will be returned (this is the default).

–**t** If –**t** is specified, **pathconv** will truncate a pathname specified in *string* in a format suitable for display as a frame title. This format is a shortened version of the full pathname, created by deleting components of the path from the middle of the string until it is under **DISPLAYW - 6** characters in length, and then inserting ellipses (**. . .**) between the remaining pieces. Ellipses are also used to show truncation at the ends of the strings if necessary, unless the –**l** option is given.

–**l** If –**l** is specified, **<** and **>** will be used instead of ellipses (**. . .**) to indicate truncation at the ends of the string generated by the –**t** option. Using –**l** allows display of the longest possible string while still notifying users it has been truncated.

–**n***num* If –**n** is specified, *num* is the maximum length of the string (in characters) generated by the –**t** option. The argument *num* can be any integer from 1 to 255.

–**v***alias* | *string*
If the –**v** option is used, then *alias* or *string* can be specified when **pathconv** is called. The argument *alias* must be an alias defined in the *alias_file* named when **fmli** was invoked. The argument *string* can only be used with the –**t** option and must be a pathname.

EXAMPLES

Here is a menu descriptor that uses **pathconv** to construct the menu title. It searches for **MYPATH** in the *alias_file* named when **fmli** was invoked:

```
menu=`pathconv –v MYPATH/ls`
 .
 .
 .
```

where there is a line in *alias_file* that defines **MYPATH**. For example, **MYPATH=$HOME/bin:/usr/bin**.

Here is a menu descriptor that takes *alias* from the standard input.

```
menu=`echo MYPATH/ls | pathconv`
 .
 .
 .
```

SEE ALSO

fmli(1)

NAME

pcmapkeys – set keyboard extended map and scancode translation for the PC console in text mode

SYNOPSIS

pcmapkeys [–**f** *mapfile* | –**n** | –**g** | –**d** | –**e**]

AVAILABILITY

x86

SUNWcsu

DESCRIPTION

pcmapkeys is a utility that permits a user to activate character mapping on input and output and keyboard extended mapping on the PC console in text mode. The keyboard extended mapping consists of the support for the deadkey and compose key sequences.

Options

–**f** *mapfile*
Installs the contents of the file *mapfile* and sets the corresponding mapping as supported by the console driver. The layout of the *mapfile* and the supported functionality are described below.

–**n** Disables and dismantles the current keyboard extended mapping. The –**f** option must be used to re-install the keyboard extended mapping.

–**g** Displays the current mappings and keyboard extended mapping (if one is installed) in hex values (see **/usr/include/sys/emap.h**). This option is mainly used for debugging purposes.

–**d** and –**e**
–**d** temporarily disables the compose key and deadkey sequences if the keyboard extended mapping is installed. The keyboard extended mapping can be enabled again by using the –**e** option (or it can be re-installed by using the –**f** option).

Consistent Keyboard-Display Mapping

The original UNIX operating system was written to support the ASCII codeset. ASCII is one of many standards to represent a number of characters internally as certain numbers. Typical for ASCII is that it supports 128 different characters, each represented by a single byte of which the 8th bit is not used. Many UNIX system applications, including the shell, took advantage of this. Starting with UNIX System V Release 3.1, most of these applications have been modified to properly support characters represented as a byte with the 8th bit set as well. This means that now 256 characters can be supported at the same time. However, a consistent coding convention needs to be applied. In the IBM PC world, an 8-bit coding referred to as IBM extended ASCII has been used for several years; MS-DOS users are quite familiar with that. In heterogeneous UNIX system environments, a different codeset, called ISO 8859, has been promoted. In both codesets, characters found in the ASCII codeset are represented in the same way. The other 128 characters are encoded differently, however, and some characters found in one codeset will be missing in the other. The Solaris for x86 system supports both codesets; actually, it supports any 8-bit one byte codeset.

To be able to use characters from the French, German, Finnish, and other alphabets, there are systems available on the market that generate 7-bit codes but display the above-mentioned characters on the screen instead of the ones found on a U.S. console. On the keyboard there are an equal number of keys, but there are different characters on the key caps. Others may support 256 different characters at a time but use their own proprietary codesets.

For example, if you are using the Solaris for x86 system with a console and a French keyboard and you do not use **pcmapkeys** to map the French keyboard tables, then if you edit a file and use the French character *é* in text, the actual code generated is ASCII 123, which is the code normally used for the left curly brace. If you look at the edited file on the console, the letter will actually appear to be a curly brace. Using **pcmapkeys** to map in the French keyboard allows consistent input and output mappings.

Input mapping

On input, any byte can be mapped to any byte. Using the example above, you could map 123 to 130, the code used for *é* in the IBM extended ASCII codeset.

Output mapping

On output, any byte can be mapped to either a byte or a string. In the above example, 130 would be mapped back to 123 to properly display the character on the screen. If the connected device is a printer that does not support the *é* character, it could be mapped to the string:

e BACKSPACE ’

Deadkeys

On typewriters, keys can be found that behave slightly differently than all the others, because when you press them, the printing wheel of the typewriter does not move. Ctrl (ˆ) and the grave accent (`) are such characters. When ` is followed by an *e*, the letter *è* is generated. This is called a deadkey or a non-spacing character. Solaris for x86 supports the use of deadkeys. Typically, the ˆ character, the ` character, and the umlaut character are used as deadkeys.

Compose sequences

Characters can also be generated using a compose sequence. A dedicated character called the ‘‘compose character’’ followed by two other keystrokes will generate a single character. As an example, COMPOSE followed by the plus and the minus sign could generate the plus / minus sign (±). Compose sequences can also be used as an alternative for deadkeys, e.g., ‘‘COMPOSE ˆ e’’ instead of ‘‘ˆ e.’’

Numeric compose sequences

Compose sequence characters that are not present on the keyboard and cannot be intuitively composed by some key sequence, for example, graphics characters, can be generated by pressing the compose key followed by three digits.

Toggle key

An optional toggle key can be defined to temporarily disable the current mapping from within an application. This can be useful when, for example, a German programmer wants easy access to the curly braces and the brackets. Use of the toggle key is analogous to the use of the –**d** and –**e** command line options.

Scancode Mapping

The keyboards of the console and some other peripherals such as SunRiver workstations behave differently than those of regular terminals. They generate what are called *scancodes* and you will also find a number of keys on these keyboards, such as the Alt key, that are not found on regular terminals. Scancodes generated by PC keyboards typically represent the location of the key on the keyboard. The keyboard driver has to properly translate these scancodes. The different national variants of a PC keyboard not only have non-English characters printed on some of the keycaps, but the order of some of the keys is different as well. Without changing the scancode translation, a French user would type **A** and see **Q** on his screen. Several status keys can influence the translated code as well. The keyboard driver, and thus the **pcmapkeys** program, makes a distinction between two sets of key combinations that can be translated.

Function keys

Up to 60 key combinations are recognized as function keys. The first 12 are the 12 function keys of a 101-key PC-keyboard (the first 10 on an 84-key keyboard).

If you do not know whether you have an 84- or 101-key keyboard, you can use the following scheme to determine which type you have:

> If your keyboard has arrow keys that are separate from the ones on the numeric keypad, then you have a 101-key keyboard.
>
> If the arrow keys on your keyboard are located on the numeric keypad only, then you have an 84-key keyboard.

F13 to F24 are the same keys used in combination with Shift, F25 to F36 when used with Ctrl, and F37 to F48 when used with Ctrl and Shift together. F49 to F60 are the keys on the numeric keypad, in the following order:

7
8
9
–
4
5
6
+
1
2
3
INS

Each of these function keys can be given a string as a value. The total length of all strings should not exceed 512 characters.

Regular keys

Scancodes generated by all keys on the PC keyboard can be translated in a different way as well. For each key, a different translation can be specified for each of the following four cases:

1. The key is pressed.
2. The key and the Shift key are pressed simultaneously.
3. The key and the Alt key are pressed simultaneously.
4. The key, the Shift, and the Alt keys are pressed simultaneously.

For each of these cases, the scancode can be translated into one of the following:

a single byte
a single byte preceded by ESC N
a single byte preceded by ESC O
a single byte preceded by ESC [

Internally, special bits are set to indicate that an escape sequence needs to be generated. Other bits are used to indicate whether the translated code should be influenced by some special keys.

Num Lock
: If the Num Lock bit is set, the regular and Shift values are swapped, as are the Alt and Shift Alt values, whenever the Num Lock LED is on. By default, only the keys on the numeric keypad have this bit set. That is why these keys generate 7, 8, 9, etc. when the Num Lock LED is on, which is the same value that would be produced if Shift were used with these keys.

Caps Lock
: This has the same effect as the Num Lock key. By default, this bit is set for all letters and not set for punctuation signs.

Ctrl
: When a key is translated into a single byte (no escape sequence) and this bit is set, the corresponding control character will be generated when the Ctrl key is pressed simultaneously. This is equally valid for the Shift, Alt, and Shift Alt combination. When this bit is not used, the Ctrl key combination will not generate anything.

mapfiles

This section describes the layout of a *mapfile* that is read by the **pcmapkeys** program.

A *mapfile* is a text file that consists of several **sections**. A sharp sign (**#**) can be used to include comments. Everything following the **#** until the end of the line will be ignored by the **pcmapkeys** program. Inside a line, C-style comments can be used as well. The beginning of each section is indicated by a *keyword*. Spaces and tabs are silently ignored and can be used at all times to improve readability. All but one section, the one that defines the *compose character*, can be left out. The order in which the different sections should appear is predefined. Here is the list of keywords in the order they should appear:

input:
toggle:
dead:
compose:
output:
scancodes:

Characters can be described in several different ways. ASCII characters can be described by putting them between single quotes. For example:

```
’a’  ’{’
```

Between single quotes, control characters can be listed by using a circumflex sign before the character that needs to be quoted. For example:

```
’^x’
```

When a backslash (\) is used, what follows will be interpreted as a decimal, octal (leading zero), or hexadecimal (leading x or X) representation of the character, although in this case the use of single quotes is not mandatory. For example:

```
’\x88’
```

is the same as:

```
0x88  (zero needed when not quoted)
```

and:

```
’\007’
```

is the same as:

```
007
```

When strings are needed, a list of character representations should be used. Quoted strings will be supported in the future.

The following paragraphs describe what goes in each section.

Input section

The input section describes which input characters should be mapped into a single byte. A very small sample input section could be:

```
input:

’A’  ’B’    # map A into B on input
’#’  0x9c   # map sharp sign into pound sign
```

Toggle section

The toggle section is a one-line section that defines which key is to toggle between mapping and no mapping. For example:

```
toggle:

’^y’        # ctrl y is the toggle key
```

Deadkey section

The deadkey section defines which keys should be treated as deadkeys. A **dead:** keyword followed by the specification of the character appears in this section for each deadkey. The subsequent lines describe what key should be generated for each key following the deadkey. A deadkey followed by a key not described in this part of the *mapfile* will not generate any key and a beep tone will be produced on the terminal. For example:

```
dead: ’^’          # circumflex is a deadkey
’ ’   ’^’          # circumflex followed by space generates circumflex
’e’   0x88         # circumflex followed by e generates e circumflex
```

```
dead: ’"’         # double quote used as a deadkey
’ ’  ’"’          # double quote space generates double quote
’a’  0x84         # double quote a generates an umlaut
```

Compose section

The first line of this section describes what the compose character is. That line should always be present in the *mapfile*. Subsequent lines consist of three character representations indicating each time that the third character needs to be generated on input when the compose character is followed by the first two. Compose sequences with the same first character should be grouped together. For example:

```
compose: ’ˆx’
’"’  ’e’   0x89        # e with umlaut is generated when typing ˆx " e
’"’  ’a’   0x84        # a with umlaut
’e’  ’"’   0x89        # e with umlaut is generated when typing ˆx e "
’a’  ’"’   0x84        # a with umlaut
```

The following example would give the wrong result. All lines starting with the same character specification should be grouped together.

```
compose: ’ˆx’
’"’  ’e’   0x89        # e with umlaut is generated when typing ˆx " e
’e’  ’"’   0x89        # e with umlaut is generated when typing ˆx e "
’"’  ’a’   0x84        # a with umlaut
’a’  ’"’   0x84# a with umlaut
```

Output section

This section describes the mapping on output, either single byte to single byte, or single byte to string. A string is specified as a series of character specifications. For example:

```
output:

0x82 ’{’                    # map e with accent to { to display e with accent
’ˆu’ ’(’’K’’I’’L’’L’’)’     # print (KILL) when kill character is used
```

Scancodes section

This section will only have an effect when your terminal is a scancode device. No error message will be produced if this section is in your *mapfile* when not needed, because the **pcmapkeys** program will find out whether the terminal is a scancode device or not. The lines in this section can have two different formats. One format will be used to describe what the values of the function keys must be. The other format describes the translation of scancodes into a byte or an escape sequence. No specific order is required.

Function keys

Here is an example of a line defining a string for a function key:

```
F13  ’d’’a’’t’’e’’\n’       # Shift F1 is the date command
```

The numbering convention of the function keys is described in a previous section. Currently, the use of quoted strings such as *"date\n"* is not supported.

Scancodes

Specifying how to translate a scancode is a more complex task. The general format of such a line is:

```
scancode normal shift alt shiftalt flags
```

scancode should list the hexadecimal representation of a scancode generated by a key (unquoted). How keys correspond with scancodes can be found in *keyboard*(7).

normal, shift, alt and **shiftalt** are character representations in one of the formats described throughout this document, optionally followed by one of the following special keywords:

|C This indicates that the key is influenced by the Ctrl key.

|N This indicates that Esc N should preceed the specified character.

|O This indicates that Esc O should preceed the specified character.

|[This indicates that Esc [should preceed the specified character.

The **normal** field defines how the scancode is translated when no other key is pressed, the **shift** field defines the translation for when the Shift key is used simultaneously, the **alt** field specifies what to do when the Alt key is pressed together with this and the **shiftalt** field contains the information on what to generate when both the Shift and Alt keys are pressed.

All five fields must be filled in. When no translation is requested (that is, the current active translation does not need to be changed) a dash (–) can be used. The sixth field is optional. This field can contain the special keyword CAPS or NUM or both, to indicate whether or not the Caps Lock key or Num Lock key status have any effect. Here is a sample line that describes the default translation for the **’Q’** key:

```
0x10 ’q’|C   ’Q’|C   ’q’|N   ’Q’|N   CAPS
```

If the normal or shift field is filled out for a scancode that represents a function key, a self-explanatory message will be produced and that translation information will be ignored.

A more detailed example of a **scancodes** section is:

```
scancodes:
# the w key
0x11 ’w’|C   ’W’|C   ’w’|N   ’W’|N   CAPS
# left square bracket and curly brace key
# control shift [ does not generate anything (no C flag)
0x1a ’[’|C   ’{’     ’[’|N   ’{’|N
# 9 on numeric keypad
0x49 ’V’|[   ’9’     ’9’|N   ’9’|N   NUM
F13 ’d’’a’’t’’e’’0 # SHIFT F1
```

More complete examples of *mapfile*s can be found in the **/usr/share/lib/keyboards** directory.

FILES

/usr/share/lib/keyboards/8859/*
sample mapfiles to be used in conjunction with ISO-8859-1 fonts (see **loadfont**(1))

/usr/share/lib/keyboards/437/*

sample mapfiles to be used in conjunction with IBM 437 fonts (see **loadfont**(1))

SEE ALSO

loadfont(1)

NOTE

The default keyboard mappings on the system are those of the ISO 8859-1 codeset. The optional IBM DOS 437 codeset is supported *only* at internationalization level 1. That is, if you choose to download keyboard mappings of the optional IBM DOS 437 codeset, there will be no support for non-standard U.S. date, time, currency, numbers, unit, and collation. There will be no support for non-English message and text presentation, and no multi-byte character support. Therefore, non-Windows users should only use IBM DOS 437 codeset in the default C locale.

NAME

pg – files perusal filter for CRTs

SYNOPSIS

pg [–*number*] [**–p** *string*] [**–cefnrs**] [**+***linenumber*] [**+/***pattern***/**] [*filename* ...]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **pg** command is a filter that allows the examination of *filenames* one screenful at a time on a CRT. If no *filename* is specified or if it encounters the file name –, **pg** reads from standard input. Each screenful is followed by a prompt. If the user types a RETURN, another page is displayed; other possibilities are listed below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

To determine terminal attributes, **pg** scans the **terminfo**(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

OPTIONS

–*number*	An integer specifying the size (in lines) of the window that **pg** is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).
–p *string*	**pg** uses *string* as the prompt. If the prompt string contains a **%d**, the first occurrence of **%d**’ in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ‘‘**:**’’.
–c	Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the **terminfo**(4) data base.
–e	**pg** does *not* pause at the end of each file.
–f	Normally, **pg** splits lines longer than the screen width, but some sequences of characters in the text being displayed (for instance, escape sequences for underlining) generate undesirable results. The **–f** option inhibits **pg** from splitting lines.
–n	Normally, commands must be terminated by a <*newline*> character. This option causes an automatic end of command as soon as a command letter is entered.
–r	Restricted mode. The shell escape is disallowed. **pg** prints an error message but does not exit.
–s	**pg** prints all messages and prompts in the standard output mode (usually inverse video).
+*linenumber*	Start up at *linenumber*.
+/*pattern***/**	Start up at the first line containing the regular expression pattern.

USAGE

Commands

The responses that may be typed when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands that cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes **pg** to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **ˆD**
Simulates scrolling half a screen forward or backward.

*i***f** Skip *i* screens of text.

*i***z** Same as <*newline*> except that *i*, if present, becomes the new default number of lines per screenful.

The following perusal commands take no *address*.

. or **ˆL** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in **ed**(1) are available. They must always be terminated by a <*newline*>, even if the –*n* option is specified.

i/*pattern*/
Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i*ˆ*pattern*ˆ
*i***?***pattern***?**
Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ˆ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, **pg** will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of **pg** can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <*newline*>, even if the –*n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q** Quit **pg**.

!*command*
Command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the –*n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally CTRL-\) or the interrupt (break) key. This causes **pg** to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, because any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then **pg** acts just like **cat**(1), except that a header is printed before each file (if there is more than one).

EXAMPLES

The following command line uses **pg** to read the system news:

```
example% news | pg –p "(Page %d):"
```

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **pg** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **pg** behaves.

LC_CTYPE
Determines how **pg** handles characters. When **LC_CTYPE** is set to a valid value, **pg** can display and handle text and filenames containing valid characters for that locale. **pg** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **pg** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

FILES

/tmp/pg∗	temporary file when input is from a pipe
/usr/share/lib/terminfo/?/∗	terminal information database

SEE ALSO

ed(1), **grep**(1), **more**(1), **terminfo**(4), **environ**(5)

NOTES

While waiting for terminal input, **pg** responds to BREAK, CTRL-C, and CTRL-\ by terminating execution. Between prompts, however, these signals interrupt **pg**'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

The terminal **/**, **^**, or **?** may be omitted from the searching commands.

If terminal tabs are not set every eight positions, undesirable results may occur.

When using **pg** as a filter with another command that changes the terminal I/O options, terminal settings may not be restored correctly.

NAME

pkginfo – display software package information

SYNOPSIS

pkginfo [**–q | –x | –l**] [**–p** | **–i**] [**–r**] [**–a** *arch*] [**–v** *version*]
[**–c** *category1*, [*category2* [, . . .]]] [*pkginst* [, *pkginst* [, . . .]]]
pkginfo [**–d** *device*] [**–R** *root_path*] [**–q | –x | –l**] [**–a** *arch*] [**–v** *version*]
[**–c** *category1* , [*category2* [, . . .]]] [*pkginst* [, *pkginst* [, . . .]]]

AVAILABILITY

SUNWcsu

DESCRIPTION

pkginfo displays information about software packages which are installed on the system (with the first synopsis) or which reside on a particular device or directory (with the second synopsis).

pkginst designates a package by its instance. An instance can be the package abbreviation or a specific instance (for example, **inst.1** or **inst.beta**). All instances of package can be requested by **inst.**∗.

Without options, **pkginfo** lists the primary category, package instance, and the names of all completely installed and partially installed packages. It displays one line for each package selected.

OPTIONS

The –**p** and –**i** options are meaningless if used in conjunction with the –**d** option.

The options –**q**, –**x**, and –**l** are mutually exclusive.

–**q**	Do not list any information. Used from a program to check whether or not a package has been installed.
–**x**	Designate an extracted listing of package information. The listing contains the package abbreviation, package name, package architecture (if available) and package version (if available).
–**l**	Specify long format, which includes all available information about the designated package(s).
–**p**	Display information for partially installed packages only.
–**i**	Display information for fully installed packages only.
–**r**	List the installation base for relocatable packages.
–**a** *arch*	Specify the architecture of the package as *arch*.
–**v** *version*	Specify the version of the package as *version*. All compatible versions can be requested by preceding the version name with a tilde (˜). Multiple white spaces are replaced with a single white space during version comparison.
–**c** *category*	Display packages that match the category *category*. Categories are defined in the category field of the **pkginfo** file. If more than one category is supplied, the package needs to match only one category in the list. The match is not case specific.

–**d** *device*	Defines a device, *device*, on which the software resides. *device* can be an absolute directory pathname or the identifiers for tape, floppy disk, removable disk, and so forth. The special token **spool** may be used to indicate the default installation spool directory (**/var/spool/pkg**).
–**R** *root_path*	Defines the full path name of a subdirectory to use as the *root_path*. All files, including package system information files, are relocated to a directory tree starting in the specified *root_path*.

SEE ALSO

pkgtrans(1), **pkgadd**(1M), **pkgask**(1M), **pkgchk**(1M), **pkgrm**(1M)

NAME

pkgmk – produce an installable package

SYNOPSIS

pkgmk [–**o**] [–**a** *arch*] [–**b** *basdir*] [–**d** *device*] [–**f** *prototype*] [–**l** *limit*]
[–**p** *pstamp*] [–**r** *rootpath*] [–**v** *version*] [*variable=value* ...] [*pkginst*]

DESCRIPTION

pkgmk produces an installable package to be used as input to the **pkgadd** command. The package contents will be in directory structure format.

The command uses the package **prototype** file as input and creates a **pkgmap** file. The contents for each entry in the **prototype** file is copied to the appropriate output location. Information concerning the contents (checksum, file size, modification date) is computed and stored in the **pkgmap** file, along with attribute information specified in the **prototype** file.

OPTIONS

–**o**	Overwrite the same instance, package instance will be overwritten if it already exists.
–**a** *arch*	Override the architecture information provided in the **pkginfo** file with *arch*.
–**b** *basdir*	Prepend the indicated *basedir* to locate relocatable objects on the source machine.
–**d** *device*	Create the package on *device*. *device* can be an absolute directory pathname or the identifiers for a floppy disk or removable disk (for example, **/dev/diskette**). The default device is the installation spool directory (**/var/spool/pkg**).
–**f** *prototype*	Use the file **prototype** as input to the command. The default **prototype** filename is **[Pp]rototype**.
–**l** *limit*	Specify the maximum size in 512 byte blocks of the output device as *limit*. By default, if the output file is a directory or a mountable device, **pkgmk** will employ the **df** command to dynamically calculate the amount of available space on the output device. This option is useful in conjunction with **pkgtrans** to create package with datastream format.
–**p** *pstamp*	Override the production stamp definition in the **pkginfo** file with *pstamp*.
–**r** *rootpath*	Ignore destination paths in the **prototype** file. Instead, use the indicated *rootpath* with the source pathname appended to locate objects on the source machine.
–**v** *version*	Override the version information provided in the **pkginfo** file with *version*.
variable=value	Place the indicated variable in the packaging environment. (See **prototype**(4) for definitions of packaging variables.)
pkginst	Specifies the package by its instance. An instance can be the package abbreviation or a specific instance (for example, **inst.1**).

SEE ALSO

pkgparam(1), **pkgproto**(1), **pkgtrans**(1)

NOTES

Architecture information is provided on the command line with the **–a** option or in the **prototype** file. If no architecture information is supplied, **pkgmk** uses the output of **uname –m**.

Version information is provided on the command line with the **–v** option or in the **prototype** file. If no version information is supplied, a default based on the current date will be provided.

Command line definitions for both architecture and version override the **prototype** definitions.

NAME

pkgparam – displays package parameter values

SYNOPSIS

pkgparam [–**v**] [–**R** *root_path*] [–**d** *device*] *pkginst* [*param* ...]
pkgparam –**f** *filename* [–**v**] [*param* ...]

AVAILABILITY

SUNWcsu

DESCRIPTION

pkgparam displays the value associated with the parameter or parameters requested on the command line. The values are located in either the **pkginfo** file for *pkginst* or from the specific file named with the –**f** option.

One parameter value is shown per line. Only the value of a parameter is given unless the –**v** option is used. With this option, the output of the command is in this format:

```
parameter1='value1'
parameter2='value2'
parameter3='value3'
```

If no parameters are specified on the command line, values for all parameters associated with the package are shown.

OPTIONS

Options and arguments for this command are:

–**v**	Verbose mode. Display name of parameter and its value.
–**R** *root_path*	Defines the full path name of a subdirectory to use as the *root_path*. All files, including package system information files, are relocated to a directory tree starting in the specified *root_path*.
–**d** *device*	Specify the *device* on which a *pkginst* is stored. It can be a directory pathname or the identifiers for tape, floppy disk or removable disk (for example, **/var/tmp**, **/dev/diskette**, and **/dev/dsk/c1d0s0**). The special token **spool** may be used to represent the default installation spool directory (**/var/spool/pkg**).
–**f** *filename*	Read *filename* for parameter values.
pkginst	Defines a specific package instance for which parameter values should be displayed.
param	Defines a specific parameter whose value should be displayed.

ERRORS

If parameter information is not available for the indicated package, the command exits with a non-zero status.

SEE ALSO

pkgtrans(1), **pkgmk**(1), **pkgparam**(1), **pkgproto**(1)

NOTES

The –**f** synopsis allows you to specify the file from which parameter values should be extracted. This file should be in the same format as a **pkginfo** file. As an example, such a file might be created during package development and used while testing software during this stage.

NAME

pkgproto – generate prototype file entries for input to pkgmk command

SYNOPSIS

pkgproto [**–i**] [**–c** *class*] [*path1*]

pkgproto [**–i**] [**–c** *class*] [*path1=path2* ...]

AVAILABILITY

SUNWcsu

DESCRIPTION

pkgproto scans the indicated paths and generates **prototype** file entries that may be used as input to the **pkgmk** command.

OPTIONS

–i	Ignores symbolic links and records the paths as ftype=f (a file) versus ftype=s(symbolic link)
–c *class*	Maps the class of all paths to *class*.
path1	Pathname where objects are located.
=path2	Pathname which should be substituted on output for *path1*.

If no paths are specified on the command line, standard input is assumed to be a list of paths. If the pathname listed on the command line is a directory, the contents of the directory is searched. However, if input is read from **stdin**, a directory specified as a pathname will not be searched.

EXAMPLES

The following two examples show uses of **pkgproto** and a partial listing of the output produced.

Example 1:

```
example% pkgproto /bin=bin /usr/bin=usrbin /etc=etc
f none bin/sed=/bin/sed 0775 bin bin
f none bin/sh=/bin/sh 0755 bin daemon
f none bin/sort=/bin/sort 0755 bin bin
f none usrbin/sdb=/usr/bin/sdb 0775 bin bin
f none usrbin/shl=/usr/bin/shl 4755 bin bin
d none etc/master.d 0755 root daemon
f none etc/master.d/kernel=/etc/master.d/kernel 0644 root daemon
f none etc/rc=/etc/rc 0744 root daemon
```

Example 2:

```
example% find / –type d –print | pkgproto
d none / 755 root root
d none /bin 755 bin bin
d none /usr 755 root root
d none /usr/bin 775 bin bin
d none /etc 755 root root
d none /tmp 777 root root
```

SEE ALSO

pkgmk(1), **pkgparam**(1), **pkgtrans**(1)

NOTES

By default, **pkgproto** creates symbolic link entries for any symbolic link encountered (ftype=s). When you use the **–i** option, **pkgproto** creates a file entry for symbolic links (ftype=f). The **prototype** file would have to be edited to assign such file types as "v" (volatile), "e" (editable), or "x" (exclusive directory). **pkgproto** detects linked files. If multiple files are linked together, the first path encountered is considered the source of the link.

By default, **pkgproto** prints prototype entries on the standard output. However, the output should be saved in a file (named **Prototype** or **prototype**, for convenience) to be used as input to the **pkgmk** command.

NAME

pkgtrans – translate package format

SYNOPSIS

pkgtrans [**–inos**] *device1 device2* [*pkginst1* [*pkginst2*] . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

pkgtrans translates an installable package from one format to another. It translates:

a file system format to a datastream

a datastream to a file system format

one file system format to another file system format

OPTIONS

The options and arguments for this command are:

–i	Copy only the **pkginfo** and **pkgmap** files.
–n	Create a new instance of the package on the destination device if any instance of this package already exists, up to the number specified by
–o	Overwrite the same instance on the destination device; package instance will be overwritten if it already exists. **MAXINST.**
–s	Indicates that the package should be written to *device2* as a datastream rather than as a file system. The default behavior is to write a file system format on devices that support both formats.
device1	Indicates the source device. The package or packages on this device will be translated and placed on *device2*.
device2	Indicates the destination device. Translated packages will be placed on this device.
pkginst	Specifies which package instance or instances on *device1* should be translated. The token **all** may be used to indicate all packages. *pkginst*.* can be used to indicate all instances of a package. If no packages are defined, a prompt shows all packages on the device and asks which to translate.

EXAMPLES

The following example translates all packages on the floppy drive **/dev/diskette** and places the translations on **/tmp**.

```
example%  pkgtrans /dev/diskette /tmp all
```

The next example translates packages **pkg1** and **pkg2** on **/tmp** and places their translations (that is, a datastream) on the **9track1** output device.

```
example%  pkgtrans /tmp 9track1 pkg1 pkg2
```

The next example translates **pkg1** and **pkg2** on **/tmp** and places them on the diskette in a datastream format.

```
example%  pkgtrans –s /tmp /dev/diskette pkg1 pkg2
```

ENVIRONMENT

The **MAXINST** variable is set in the pkginfo file and declares the maximum number of package instances.

SEE ALSO

pkginfo(1), **pkgmk**(1), **pkgparam**(1), **pkgproto**(1), **installf**(1M), **pkgadd**(1M), **pkgask**(1M), **pkgrm**(1M), **removef**(1M)

Application Packaging Developer's Guide

NOTES

Device specifications can be either the special node name (for example, **/dev/diskette**) or a device alias (for example, **diskette1**). The device **spool** indicates the default spool directory. Source and destination devices cannot be the same.

By default, **pkgtrans** will not translate any instance of a package if any instance of that package already exists on the destination device. Using the –**n** option creates a new instance if an instance of this package already exists. Using the –**o** option overwrites an instance of this package if it already exists. Neither of these options are useful if the destination device is a datastream.

NAME

plot, aedplot, atoplot, bgplot, crtplot, dumbplot, gigiplot, hpplot, implot, plottoa, t300, t300s, t4013, t450, tek, vplot, hp7221plot – graphics filters for various plotters

SYNOPSIS

/usr/ucb/plot [**–T***terminal*]

AVAILABILITY

SUNWscpu

DESCRIPTION

plot reads plotting instructions (see **plot**(4B)) from the standard input and produces plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* is specified, the environment variable **TERM** is used. The default *terminal* is **tek**.

ENVIRONMENT

Except for **ver**, the following terminal-types can be used with ‘**lpr –g**’ (see **lpr**) to produce plotted output:

2648 | **2648a** | **h8** | **hp2648** | **hp2648a**	Hewlett Packard® 2648 graphics terminal.
hp7221 | **hp7** | **h7** |	Hewlett Packard® 7221 plotter.
300	DASI 300 or GSI terminal (Diablo® mechanism).
300s | **300S**	DASI 300s terminal (Diablo® mechanism).
450	DASI Hyterm 450 terminal (Diablo® mechanism).
4013	Tektronix® 4013 storage scope.
4014 | **tek**	Tektronix 4014 and 4015 storage scope with Enhanced Graphics Module. (Use 4013 for Tektronix® 4014 or 4015 without the Enhanced Graphics Module).
aed	AED 512 color graphics terminal.
bgplot | **bitgraph**	BBN bitgraph graphics terminal.
crt	Any crt terminal capable of running **vi**(1).
dumb | **un** | **unknown**	Dumb terminals without cursor addressing or line printers.
gigi | **vt125**	DEC® vt125 terminal.
h7 | **hp7** | **hp7221**	Hewlett Packard® 7221 graphics terminal.
implot	Imagen plotter.
var	Benson Varian printer-plotter
ver	Versatec® D1200A printer-plotter. The output is scan-converted and suitable input to ‘**lpr –v**’.

FILES

/usr/ucb/aedplot
/usr/ucb/atoplot
/usr/ucb/bgplot
/usr/ucb/crtplot
/usr/ucb/dumbplot
/usr/ucb/gigiplot
/usr/ucb/hp7221plot
/usr/ucb/hpplot
/usr/ucb/implot
/usr/ucb/plot
/usr/ucb/plottoa
/usr/ucb/t300
/usr/ucb/t300s
/usr/ucb/t4013
/usr/ucb/t450
/usr/ucb/tek
/usr/ucb/vplot
/var/ucb/vplot*nnnnnn*

SEE ALSO

graph(1), **vi**(1), **lpr**(1B), **plot**(4B)

NAME

postdaisy – PostScript translator for Diablo 630 daisy-wheel files

SYNOPSIS

postdaisy [–**c** *num*] [–**f** *num*] [–**h** *num*] [–**m** *num*] [–**n** *num*] [–**o** *list*]
[–**p** *mode*] [–**r** *num*] [–**s** *num*] [–**v** *num*] [–**x** *num*] [–**y** *num*] [*file* . . .]

/usr/lib/lp/postscript/postdaisy

DESCRIPTION

The **postdaisy** filter translates Diablo 630 daisy-wheel *files* into PostScript and writes the results on the standard output. If no *files* are specified, or if – is one of the input *files*, the standard input is read.

OPTIONS

–**c** *num*	Print *num* copies of each page. By default only one copy is printed.
–**f** *name*	Print *files* using font *name.* Any PostScript font can be used, although the best results will be obtained only with constant-width fonts. The default font is Courier.
–**h** *num*	Set the initial horizontal motion index to *num.* Determines the character advance and the default point size, unless the –**s** option is used. The default is **12.**
–**m** *num*	Magnify each logical page by the factor *num.* Pages are scaled uniformly about the origin, which is located near the upper left corner of each page. The default magnification is **1.0.**
–**n** *num*	Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default, *num* is set to **1.**
–**o** *list*	Print pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1* – *N2.* A missing *N1* means the lowest numbered page, a missing *N2* means the highest. The page range is an expression of logical pages rather than physical sheets of paper. For example, if you are printing two logical pages to a sheet, and you specified a range of **4,** then two sheets of paper would print, containing four page layouts. If you specified a page range of **3-4,** when requesting two logical pages to a sheet; then *only* page 3 and page 4 layouts would print, and they would appear on one physical sheet of paper.
–**p** *mode*	Print *files* in either portrait or landscape *mode.* Only the first character of *mode* is significant. The default *mode* is portrait.
–**r** *num*	Selects carriage return and line feed behavior. If *num* is 1, a line feed generates a carriage return. If *num* is 2, a carriage return generates a line feed. Setting *num* to 3 enables both modes.
–**s** *num*	Use point size *num* instead of the default value set by the initial horizontal motion index.
–**v** *num*	Set the initial vertical motion index to *num.* The default is **8.**
–**x** *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed near the upper left corner of the page, with positive x to the right and positive y down the page. Positive *num* moves

everything right. The default offset is **0.25** inches.

–y *num* Translate the origin *num* inches along the positive y axis. Positive *num* moves text up the page. The default offset is **–0.25** inches.

FILES

/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/ps.requests

SEE ALSO

download(1), **dpost**(1), **postdmd**(1), **postio**(1), **postmd**(1), **postprint**(1), **postreverse**(1), **posttek**(1)

DIAGNOSTICS

An exit status of **0** is returned if *files* were successfully processed.

NAME

postdmd – PostScript translator for DMD bitmap files

SYNOPSIS

postdmd [–**b** *num*] [–**c** *num*] [–**f**] [–**m** *num*] [–**n** *num*] [–**o** *list*] [–**p** *mode*]
[–**x** *num*] [–**y** *num*] [*file* . . .]

/usr/lib/lp/postscript/postdmd

DESCRIPTION

postdmd translates DMD bitmap *files*, as produced by *dmdps*, or *files* written in the Ninth Edition **bitfile**(9.5) format into PostScript and writes the results on the standard output. If no *files* are specified, or if – is one of the input *files*, the standard input is read.

OPTIONS

–**b** *num*	Pack the bitmap in the output file using *num* byte patterns. A value of **0** turns off all packing of the output file. By default, *num* is **6.**
–**c** *num*	Print *num* copies of each page. By default only one copy is printed.
–**f**	Flip the sense of the bits in *files* before printing the bitmaps.
–**m** *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, which by default is located at the center of each page. The default magnification is **1.0.**
–**n** *num*	Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default *num* is set to **1.**
–**o** *list*	Print pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1* – *N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest. The page range is an expression of logical pages rather than physical sheets of paper. For example, if you are printing two logical pages to a sheet, and you specified a range of **4,** then two sheets of paper would print, containing four page layouts. If you specified a page range of **3-4,** when requesting two logical pages to a sheet; then *only* page 3 and page 4 layouts would print, and they would appear on one physical sheet of paper.
–**p** *mode*	Print *files* in either portrait or landscape *mode*. Only the first character of *mode* is significant. The default *mode* is portrait.
–**x** *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up the page. Positive *num* moves everything right. The default offset is **0** inches.
–**y** *num*	Translate the origin *num* inches along the positive y axis. Positive *num* moves everything up the page. The default offset is **0.**

Only one bitmap is printed on each logical page, and each of the input *files* must contain complete descriptions of at least one bitmap. Decreasing the pattern size using the –**b** option may help throughput on printers with fast processors (such as PS-810s), while increasing the pattern size will often be the right move on older models (such as PS-800s).

FILES

/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/ps.requests

SEE ALSO

download(1), **dpost**(1), **postdaisy**(1), **postio**(1), **postmd**(1), **postprint**(1), **postreverse**(1), **posttek**(1)

DIAGNOSTICS

An exit status of **0** is returned if *files* were successfully processed.

NAME

postio – serial interface for PostScript printers

SYNOPSIS

postio **–l** *line* [**–D**] [**–i**] [**–q**] [**–t**] [**–S**] [**–b** *speed*] [**–B** *num*] [**–L** *file*]
[**–P** *string*] [**–R** *num*] [*file* . . .]

/usr/lib/lp/postscript/postio

DESCRIPTION

postio sends *files* to the PostScript printer attached to *line.* If no *files* are specified the standard input is sent.

OPTIONS

The first group of *options* should be sufficient for most applications:

–D	Enable debug mode. Guarantees that everything read on *line* will be added to the log file (standard error by default).
–q	Prevents status queries while *files* are being sent to the printer. When status queries are disabled a dummy message is appended to the log file before each block is transmitted.
–b *speed*	Transmit data over *line* at baud rate *speed.* Recognized baud rates are 1200, 2400, 4800, 9600, and 19200. The default *speed* is **9600** baud.
–B *num*	Set the internal buffer size for reading and writing *files* to *num* bytes. By default *num* is **2048** bytes.
–l *line*	Connect to the printer attached to *line.* In most cases there is no default and **postio** must be able to read and write *line*. If the *line* does not begin with a **/** it may be treated as a Datakit destination.
–L *file*	Data received on *line* gets put in *file*. The default log *file* is standard error. Printer or status messages that don't show a change in state are not normally written to *file* but can be forced out using the **–D** option.
–P *string*	Send *string* to the printer before any of the input files. The default *string* is simple PostScript code that disables timeouts.
–R *num*	Run *postio* as a single process if *num* is 1 or as separate read and write processes if *num* is 2. By default **postio** runs as a single process.

The next two *options* are provided for users who expect to run **postio** on their own. Neither is suitable for use in spooler interface programs:

–i	Run the program in interactive mode. Any *files* are sent first and followed by the standard input. Forces separate read and write processes and overrides many other options. To exit interactive mode use your interrupt or quit character. To get a friendly interactive connection with the printer type **executive** on a line by itself.
–t	Data received on *line* and not recognized as printer or status information is written to the standard output. Forces separate read and write processes. Convenient if you have a PostScript program that will be returning useful data to the host.

The last option is not generally recommended and should only be used if all else fails to provide a reliable connection:

–S	Slow the transmission of data to the printer. Severely limits throughput, runs as a single process, disables the **–q** option, limits the internal buffer size to 1024 bytes, can use an excessive amount of CPU time, and does nothing in interactive mode.

The best performance will usually be obtained by using a large internal buffer (the **–B** option) and by running the program as separate read and write processes (the **–R 2** option). Inability to fork the additional process causes **postio** to continue as a single read / write process. When one process is used, only data sent to the printer is flow controlled.

The *options* are not all mutually exclusive. The **–i** option always wins, selecting its own settings for whatever is needed to run interactive mode, independent of anything else found on the command line. Interactive mode runs as separate read and write processes and few of the other *options* accomplish anything in the presence of the **–i** option. The **–t** option needs a reliable two way connection to the printer and therefore tries to force separate read and write processes. The **–S** option relies on the status query mechanism, so **–q** is disabled and the program runs as a single process.

In most cases **postio** starts by making a connection to *line* and then attempts to force the printer into the IDLE state by sending an appropriate sequence of **ˆT** (status query), **ˆC** (interrupt), and **ˆD** (end of job) characters. When the printer goes IDLE, *files* are transmitted along with an occasional **ˆT** (unless the **–q** option was used). After all the *files* are sent the program waits until it's reasonably sure the job is complete. Printer generated error messages received at any time except while establishing the initial connection (or when running interactive mode) cause **postio** to exit with a non-zero status. In addition to being added to the log file, printer error messages are also echoed to standard error.

EXAMPLES

Run as a single process at 9600 baud and send *file1* and *file2* to the printer attached to **/dev/tty01**:

```
example% postio –l /dev/tty01 file1 file2
```

Same as above except two processes are used, the internal buffer is set to 4096 bytes, and data returned by the printer gets put in file *log*:

```
example% postio –R 2 –B 4096 –l/dev/tty01 –L log file1 file2
```

Establish an interactive connection with the printer at Datakit destination *my/printer*:

```
example% postio –i –l my/printer
```

Send file program to the printer connected to **/dev/tty22**, recover any data in file results, and put log messages in file *log*:

```
example% postio –t –l /dev/tty22 –L log program >results
```

SEE ALSO

download(1), **dpost**(1), **postdaisy**(1), **postdmd**(1), **postmd**(1), **postprint**(1), **postreverse**(1), **posttek**(1)

DIAGNOSTICS

An exit status of **0** is returned if the files ran successfully. System errors (such as an inability to open the line) set the low order bit in the exit status, while PostScript errors set bit 1. An exit status of **2** usually means the printer detected a PostScript error in the input *files*.

NOTES

The input *files* are handled as a single PostScript job. Sending several different jobs, each with their own internal end of job mark (**ˆD**) is not guaranteed to work properly. **postio** may quit before all the jobs have completed and could be restarted before the last one finishes.

All the capabilities described above may not be available on every machine or even across the different versions of the UNIX system that are currently supported by the program.

There may be no default *line*, so using the **–l** option is strongly recommended. If omitted, **postio** may attempt to connect to the printer using the standard output. If Datakit is involved, the **–b** option may be ineffective and attempts by **postio** to impose flow control over data in both directions may not work. The **–q** option can help if the printer is connected to RADIAN. The **–S** option is not generally recommended and should be used only if all other attempts to establish a reliable connection fail.

NAME

postmd – matrix display program for PostScript printers

SYNOPSIS

postmd [–**b** *num*] [–**c** *num*] [–**d** *dimen*] [–**g** *list*] [–**i** *list*] [–**m** *num*] [–**n** *num*]
[–**o** *list*] [–**p** *mode*] [–**w** *window*] [–**x** *num*] [–**y** *num*] [*file* . . .]

/usr/lib/lp/postscript/postmd

DESCRIPTION

The **postmd** filter reads a series of floating point numbers from *files*, translates them into a PostScript gray scale image, and writes the results on the standard output. In a typical application the numbers might be the elements of a large matrix, written in row major order, while the printed image could help locate patterns in the matrix. If no *files* are specified, or if – is one of the input *files*, the standard input is read.

OPTIONS

–**b** *num*	Pack the bitmap in the output file using *num* byte patterns. A value of 0 turns off all packing of the output file. By default, *num* is **6.**
–**c** *num*	Print *num* copies of each page. By default, only one copy is printed.
–**d** *dimen*	Sets the default matrix dimensions for all input *files* to *dimen*. The *dimen* string can be given as rows or rows**x**columns. If *columns* is omitted it will be set to rows. By default, **postmd** assumes each matrix is square and sets the number of rows and columns to the square root of the number of elements in each input file.
–**g** *list*	*list* is a comma or space separated string of integers, each lying between 0 and 255 inclusive, that assigns PostScript gray scales to the regions of the real line selected by the –**i** option. 255 corresponds to white, and 0, to black. The **postmd** filter assigns a default gray scale that omits white (that is, 255) and gets darker as the regions move from left to right along the real line.
–**i** *list*	*list* is a comma, space or slash(/) separated string of *N* floating point numbers that partition the real line into 2*N*+1 regions. The *list* must be given in increasing numerical order. The partitions are used to map floating point numbers read from the input *files* into gray scale integers that are either assigned automatically by **postmd** or arbitrarily selected using the –**g** option. The default interval *list* is –**1,0,1**, which partions the real line into seven regions.
–**m** *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin which, by default, is located at the center of each page. The default magnification is **1.0.**
–**n** *num*	Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default, *num* is set to **1.**
–**o** *list*	Print pages whose numbers are given in the comma separated *list*. The list contains single numbers *N* and ranges *N1* – *N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest. The page range is an expression of logical pages rather than physical sheets of paper. For example, if you are printing two logical pages to a sheet, and you specified a range of **4,** then two sheets of paper would print, containing four page layouts. If you

specified a page range of **3-4,** when requesting two logical pages to a sheet; then *only* page 3 and page 4 layouts would print, and they would appear on one physical sheet of paper.

−p *mode* Print *files* in either portrait or landscape *mode.* Only the first character of *mode* is significant. The default *mode* is portrait.

−w *window*
Window is a comma or space separated list of four positive integers that select the upper left and lower right corners of a submatrix from each of the input *files*. Row and column indices start at 1 in the upper left corner and the numbers in the input *files* are assumed to be written in row major order. By default, the entire matrix is displayed.

−x *num* Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up the page. Positive *num* moves everything right. The default offset is **0** inches.

−y *num* Translate the origin *num* inches along the positive y axis. Positive *num* moves everything up the page. The default offset is **0.**

Only one matrix is displayed on each logical page, and each of the input *files* must contain complete descriptions of exactly one matrix. Matrix elements are floating point numbers arranged in row major order in each input file. White space, including newlines, is not used to determine matrix dimensions. By default, **postmd** assumes each matrix is square and sets the number of rows and columns to the square root of the number of elements in the input file. Supplying default dimensions on the command line with the −**d** option overrides this default behavior, and in that case the dimensions apply to all input *files*.

An optional header can be supplied with each input file and is used to set the matrix dimensions, the partition of the real line, the gray scale map, and a window into the matrix. The header consists of keyword / value pairs, each on a separate line. It begins on the first line of each input file and ends with the first unrecognized string, which should be the first matrix element. Values set in the header take precedence, but apply only to the current input file. Recognized header keywords are **dimension**, **interval**, **grayscale**, and **window**. The syntax of the value string that follows each keyword parallels what is accepted by the −**d**, −**i**, −**g**, and −**w** options.

EXAMPLES

For example, suppose file initially contains the 1000 numbers in a 20x50 matrix. Then you can produce exactly the same output by completing three steps. First, issue the following command line:

```
example% postmd −d20x50 −i"−100 100" −g0,128,254,128,0 file
```

Second, prepend the following header to *file*:

```
dimension 20x50
interval  −100.0 .100e+3
grayscale 0 128 254 128 0
```

Third, issue the following command line:

```
example% postmd file
```

The interval list partitions the real line into five regions and the gray scale list maps numbers less than –100 or greater than 100 into 0 (that is, black), numbers equal to –100 or 100 into 128 (that is, 50 percent black), and numbers between –100 and 100 into 254 (that is, almost white).

FILES

/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/ps.requests

SEE ALSO

dpost(1), **postdaisy**(1), **postdmd**(1), **postio**(1), **postprint**(1), **postreverse**(1), **posttek**(1)

DIAGNOSTICS

An exit status of **0** is returned if *files* were successfully processed.

NOTES

The largest matrix that can be adequately displayed is a function of the interval and gray scale lists, the printer resolution, and the paper size. A 600x600 matrix is an optimistic upper bound for a two element interval list (that is, five regions) using 8.5x11 inch paper on a 300 dpi printer.

Using white (that is, 255) in a gray scale list is not recommended and won’t show up in the legend and bar graph that **postmd** displays below each image.

NAME

postplot – PostScript translator for plot(4) graphics files

SYNOPSIS

postplot [**–c** *num*] [**–f** *name*] [**–m** *num*] [**–n** *num*] [**–o** *list*] [**–p** *mode*] [**–w** *num*] [**–x** *num*] [**–y** *num*] [*filename* . . .]

/usr/lib/lp/postscript/postplot

DESCRIPTION

The **postplot** filter translates **plot**(1B) graphics *filenames* into PostScript and writes the results on the standard output. If no *filenames* are specified, or if – is one of the input *filenames*, the standard input is read.

OPTIONS

–c *num*	Print *num* copies of each page. By default, only one copy is printed.
–f *name*	Print text using font *name*. Any PostScript font can be used, although the best results will be obtained only with constant width fonts. The default font is Courier.
–m *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin which, by default, is located at the center of each page. The default magnification is 1.0.
–n *num*	Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default, *num* is set to 1.
–o *list*	Print pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1* – *N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest.
–p *mode*	Print *filenames* in either portrait or landscape *mode*. Only the first character of *mode* is significant. The default *mode* is landscape.
–w *num*	Set the line width used for graphics to *num* points, where a point is approximately 1 / 72 of an inch. By default, *num* is set to 0 points, which forces lines to be one pixel wide.
–x *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up the page. Positive *num* moves everything right. The default offset is 0.0 inches.
–y *num*	Translate the origin *num* inches along the positive y axis. Positive *num* moves everything up the page. The default offset is 0.0.

FILES

/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/postplot.ps
/usr/lib/lp/postscript/ps.requests

SEE ALSO

download(1), **dpost**(1), **plot**(1B), **postdaisy**(1), **postdmd**(1), **postio**(1), **postmd**(1), **postprint**(1), **postreverse**(1)

DIAGNOSTICS

An exit status of 0 is returned if *filenames* were successfully processed.

NOTES

The default line width is too small for write-white print engines, such as the one used by the PS-2400.

NAME

postprint – PostScript translator for text files

SYNOPSIS

postprint [–**c** *num*] [–**f** *name*] [–**l** *num*] [–**m** *num*] [–**n** *num*] [–**o** *list*]
[–**p** *mode*] [–**r** *num*] [–**s** *num*] [–**t** *num*] [–**x** *num*] [–**y** *num*]
[*file*. . .]

/usr/lib/lp/postscript/postprint

DESCRIPTION

The **postprint** filter translates text *files* into PostScript and writes the results on the standard output. If no *files* are specified, or if – is one of the input *files*, the standard input is read.

OPTIONS

–**c** *num*	Print *num* copies of each page. By default, only one copy is printed.
–**f** *name*	Print *files* using font *name*. Any PostScript font can be used, although the best results will be obtained only with constant width fonts. The default font is Courier.
–**l** *num*	Set the length of a page to *num* lines. By default, *num* is **66.** Setting *num* to **0** is allowed, and will cause *postprint* to guess a value, based on the point size that's being used.
–**m** *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, which is located near the upper left corner of each page. The default magnification is **1.0.**
–**n** *num*	Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default, *num* is set to **1.**
–**o** *list*	Print pages whose numbers are given in the comma-separated *list*. The *list* contains single numbers *N* and ranges *N1* – *N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest. The page range is an expression of logical pages rather than physical sheets of paper. For example, if you are printing two logical pages to a sheet, and you specified a range of **4,** then two sheets of paper would print, containing four page layouts. If you specified a page range of **3-4,** when requesting two logical pages to a sheet; then *only* page 3 and page 4 layouts would print, and they would appear on one physical sheet of paper.
–**p** *mode*	Print *files* in either portrait or landscape *mode*. Only the first character of *mode* is significant. The default *mode* is portrait.
–**r** *num*	Selects carriage return behavior. Carriage returns are ignored if *num* is **0,** cause a return to column 1 if *num* is **1,** and generate a newline if *num* is **2.** The default *num* is **0.**
–**s** *num*	Print *files* using point size *num*. When printing in landscape mode *num* is scaled by a factor that depends on the imaging area of the device. The default size for portrait mode is **10.** Note that increasing point size increases virtual image size, so you either need to load larger paper, or use the –**l0** option to scale the number of lines per page.

–t *num* Assume tabs are set every *num* columns, starting with the first column. By default, tabs are set every **8** columns.

–x *num* Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed near the upper left corner of the page, with positive x to the right and positive y down the page. Positive *num* moves everything to the right. The default offset is **0.25** inches.

–y *num* Translate the origin *num* inches along the positive y axis. Positive *num* moves text up the page. The default offset is –**0.25** inches.

A new logical page is started after 66 lines have been printed on the current page, or whenever an ASCII form feed character is read. The number of lines per page can be changed using the **–l** option. Unprintable ASCII characters are ignored, and lines that are too long are silently truncated by the printer.

EXAMPLES

To print *file1* and *file2* in landscape mode, issue the following command:

```
example% postprint –pland file1 file2
```

To print three logical pages on each physical page in portrait mode:

```
example% postprint –n3 file
```

FILES

/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/ps.requests

SEE ALSO

download(1), **dpost**(1), **postdaisy**(1), **postdmd**(1), **postio**(1), **postmd**(1), **postreverse**(1), **posttek**(1)

DIAGNOSTICS

An exit status of **0** is returned if *files* were successfully processed.

NAME

postreverse – reverse the page order in a PostScript file

SYNOPSIS

postreverse [**–o** *list*] [**–r**] [*file*]

/usr/lib/lp/postscript/postreverse

DESCRIPTION

The **postreverse** filter reverses the page order in files that conform to Adobe's Version 1.0 or Version 2.0 file structuring conventions, and writes the results on the standard output. Only one input *file* is allowed and if no *file* is specified, the standard input is read.

The **postreverse** filter can handle a limited class of files that violate page independence, provided all global definitions are bracketed by **%%BeginGlobal** and **%%EndGlobal** comments. In addition, files that mark the end of each page with **%%EndPage: label ordinal** comments will also reverse properly, provided the prologue and trailer sections can be located. If **postreverse** fails to find an **%%EndProlog** or **%%EndSetup** comment, the entire *file* is copied, unmodified, to the standard output.

Because global definitions are extracted from individual pages and put in the prologue, the output file can be minimally conforming, even if the input *file* was not.

OPTIONS

–o *list* Select pages whose numbers are given in the comma-separated *list*. The *list* contains single numbers *N* and ranges *N1 – N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest. The page range is an expression of logical pages rather than physical sheets of paper. For example, if you are printing two logical pages to a sheet, and you specified a range of **4,** then two sheets of paper would print, containing four page layouts. If you specified a page range of **3-4,** when requesting two logical pages to a sheet; then *only* page 3 and page 4 layouts would print, and they would appear on one physical sheet of paper.

–r Do not reverse the pages in *file*.

EXAMPLES

To select pages 1 to 100 from *file* and reverse the pages:

```
example% postreverse –o1–100 file
```

To print four logical pages on each physical page and reverse all the pages:

```
example% postprint –n4 file | postreverse
```

To produce a minimally conforming file from output generated by **dpost** without reversing the pages:

```
example% dpost file | postreverse –r
```

SEE ALSO

download(1), **dpost**(1), **postdaisy**(1), **postdmd**(1), **postio**(1), **postmd**(1), **postprint**(1), **posttek**(1)

DIAGNOSTICS

An exit status of **0** is returned if *file* was successfully processed.

NOTES

No attempt has been made to deal with redefinitions of global variables or procedures. If standard input is used, the input *file* will be read three times before being reversed.

NAME

posttek – PostScript translator for Tektronix 4014 files

SYNOPSIS

posttek [**–c** *num*] [**–f** *name*] [**–m** *num*] [**–n** *num*] [**–o** *list*] [**–p** *mode*] [**–w** *num*]
[**–x** *num*] [**–y** *num*] [*file* . . .]

/usr/lib/lp/postscript/posttek

DESCRIPTION

The **posttek** filter translates Tektronix 4014 graphics *files* into PostScript and writes the results on the standard output. If no *files* are specified, or if – is one of the input *files*, the standard input is read.

OPTIONS

–c *num*	Print *num* copies of each page. By default, only one copy is printed.
–f *name*	Print text using font *name*. Any PostScript font can be used, although the best results will be obtained only with constant width fonts. The default font is Courier.
–m *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin which, by default, is located at the center of each page. The default magnification is **1.0.**
–n *num*	Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default, *num* is set to **1.**
–o *list*	Print pages whose numbers are given in the comma-separated *list*. The *list* contains single numbers *N* and ranges *N1* – *N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest. The page range is an expression of logical pages rather than physical sheets of paper. For example, if you are printing two logical pages to a sheet, and you specified a range of **4,** then two sheets of paper would print, containing four page layouts. If you specified a page range of **3-4,** when requesting two logical pages to a sheet; then *only* page 3 and page 4 layouts would print, and they would appear on one physical sheet of paper.
–p *mode*	Print *files* in either portrait or landscape *mode*. Only the first character of *mode* is significant. The default *mode* is landscape.
–w *num*	Set the line width used for graphics to *num* points, where a point is approximately 1/72 of an inch. By default, *num* is set to **0** points, which forces lines to be one pixel wide.
–x *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up the page. Positive *num* moves everything right. The default offset is **0.0** inches.
–y *num*	Translate the origin *num* inches along the positive y axis. Positive *num* moves everything up the page. The default offset is **0.0.**

FILES

/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/ps.requests

SEE ALSO

download(1), **dpost**(1), **postdaisy**(1), **postdmd**(1), **postio**(1), **postmd**(1), **postprint**(1), **postreverse**(1)

DIAGNOSTICS

An exit status of **0** is returned if *files* were successfully processed.

NOTES

The default line width is too small for write-white print engines, such as the one used by the PS-2400.

NAME

pr – print files

SYNOPSIS

pr [[–*columns*] [–**w***width*] [–**a**]] [–**e***ck*] [–**i***ck*] [–**drtfp**] [**+page**] [–**n***ck*] [–**o***offset*] [–**l***length*]
[–**s***separator*] [–**h***header*] [–**F**] [*filename* . . .]

pr [[–**m**] [–**w***width*]] [–**e***ck*] [–**i***ck*] [–**drtfp**] [**+page**] [–**n***ck*] [–**o***offset*] [–**l***length*] [–**s***separator*]
[–**h***header*] [–**F**] [*filename1 filename2* . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **pr** command formats and prints the contents of a file. If *filename* is –, or if no files are specified, **pr** assumes standard input. **pr** prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, the date and time that the file was last modified, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with –**h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the –**s** option is used, lines are not truncated and columns are separated by the *separator* character.

Either –*columns* or –**m** should be used to produce multi-column output. –**a** should only be used with –*columns* and not –**m**.

OPTIONS

+*page*

Begin printing with page numbered *page* (default is 1).

–*columns*

Print *columns* columns of output (default is 1). For example **pr** –*3* produces the following output:

```
Print    of     in
the      one    three
lines    file   columns.
```

Columns are not balanced; if, for example, there are as many lines in the file as there are lines on the page, only one column will be printed.

Output appears as if –**e** and –**i** are on for multi-column output. May not use with –**m**.

–a

Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.

–m

Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with –*column*.

−**d**	Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.
−**e***ck*	Expand input tabs to character positions *k*+1, 2∗*k*+1, 3∗*k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
−**i***ck*	In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2∗*k*+1, 3∗*k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).
−**n***ck*	Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of −**m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
−**w***width*	Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (−*column* and −**m**). There is no line limit for single column output.
−**o***offset*	Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
−**l***length*	Set the length of a page to *length* lines (default is 66). A *length* of 0 specifies the default length. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When −**l***length* is used and *length* exceeds 10, then *length*−10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text; see the −**t** option.
−**h** *header*	Use *header* as the text line of the header to be printed instead of the file name. −**h** is ignored when −**t** is specified or −**l***length* is specified and the value of *length* is 10 or less. (−**h** is the only **pr** option requiring space between the option and argument.)
−**p**	Pause before beginning each page if the output device is a terminal. **pr** rings the terminal bell and waits for a carriage return.
−**f**	Use a single form-feed character for new pages (default is to use a sequence of line feeds). Pause before beginning the first page if the standard output is associated with a terminal.
−**r**	Print no diagnostic reports on files that cannot be opened.

–t	Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **–t** overrides the **–h** option.
–s*separator*	Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless **–w** is specified.
–F	Fold the lines of the input file. When used in multi-column mode (with the **–a** or **–m** options) lines will be folded to fit the current column's width, otherwise they will be folded to fit the current line width (80 columns).

EXAMPLES

Print **filename1** and **filename2** as a double-spaced, three-column listing headed by ‘‘**file list**’’:

```
example% pr –3dh "file list" filename1 filename2
```

Copy **filename1** to **filename2**, expanding tabs to columns 10, 19, 28, 37, . . . :

```
example% pr –e9 –t < filename1 > filename2
```

Print **filename1** and **filename2** simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

```
example% pr –t –n filename1 | pr –t –m –n filename2 –
```

ENVIRONMENT

If any of the **LC_∗** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **pr** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_∗** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **pr** behaves.

LC_CTYPE

Determines how **pr** handles characters. When **LC_CTYPE** is set to a valid value, **pr** can display and handle text and filenames containing valid characters for that locale. **pr** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **pr** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME

Determines how **pr** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/dev/tty∗	If standard output is directed to one of the special files **/dev/tty**∗, then other output directed to this terminal is delayed until standard output is completed. This prevents error messages from being interspersed throughout the output.

SEE ALSO

cat(1), **fold**(1), **more**(1), **pg**(1), **environ**(5)

NAME

print – shell built-in function to output characters to the screen or window

SYNOPSIS

ksh

print [–**Rnprsu**[*n*]] [*arg* . . .]

DESCRIPTION

ksh

The shell output mechanism. With no flags or with flag – or – –, the arguments are printed on standard output as described by **echo**(1). The exit status is 0, unless the output file is not open for writing.

–**n**	suppresses **new-line** from being added to the output.
–**R** –**r**	(raw mode) ignore the escape conventions of **echo**. The –**R** option will print all subsequent arguments and options other than –**n**.
–**p**	causes the arguments to be written onto the pipe of the process spawned with | **&** instead of standard output.
–**s**	causes the arguments to be written onto the history file instead of standard output.
–**u** [*n*]	flag can be used to specify a one digit file descriptor unit number *n* on which the output will be placed. The default is 1.

SEE ALSO

echo(1), **ksh**(1)

NAME

printenv – display environment variables currently set

SYNOPSIS

/usr/ucb/printenv [*variable*]

AVAILABILITY

SUNWscpu

DESCRIPTION

printenv prints out the values of the variables in the environment. If a *variable* is specified, only its value is printed.

SEE ALSO

csh(1), **echo**(1), **sh**(1), **stty**(1), **tset**(1B), **environ**(5)

DIAGNOSTICS

If a *variable* is specified and it is not defined in the environment, **printenv** returns an exit status of **1**.

NAME

printf – print formatted output

SYNOPSIS

printf *format* [*argument*. . .]

AVAILABILITY

SUNWloc

DESCRIPTION

The **printf** command converts, formats, and prints its *argument*s under control of the *format*. It fully supports conversion specifications for strings (**%s** descriptor); however, the results are undefined for the other conversion specifications supported by **printf**(3S).

OPTIONS

format	A character string that contains three types of objects: 1) plain characters, which are simply copied to the output stream; 2) conversion specifications, each of which results in fetching zero or more *argument*s; and 3) C-language escape sequences, which are translated into the corresponding characters.
argument	String(s) to be printed under the control of *format*. The results are undefined if there are insufficient *argument*s for the format. If the format is exhausted while *argument*s remain, the excess *argument*s are simply ignored.

Each conversion specification is introduced by the character **%**. After the **%**, the following appear in sequence:

> An optional field, consisting of a decimal digit string followed by a **$**, specifying the next *argument* to be converted. If this field is not provided, the *argument* following the last *argument* converted is used.
>
> An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it is padded on the left (or right, if the left-adjustment flag '–' has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.
>
> An optional *precision* that gives the maximum number of characters to be printed from a string in **%s** conversion. The precision takes the form of a period (**.**) followed by a decimal digit string; a null digit string is treated as zero (nothing is printed). Padding specified by the precision overrides the padding specified by the field width. That is, if *precision* is specified, its value is used to control the number of characters printed.
>
> A field width or precision or both may be indicated by an asterisk (∗) instead of a digit string. In this case, an integer *argument* supplies the field width or precision. The *argument* that is actually converted is not fetched until the conversion letter is seen, so the *argument*s specifying field width or precision must appear *before* the *argument* (if any) to be converted. A negative field width argument is taken as a **'–'** (left-adjustment) flag followed by a positive field width. If the precision argument is negative, it is changed to zero (nothing is printed). In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

The conversion characters and their meanings are:

%s The *argument* is taken to be a string and characters from the string are printed until a null character (**0**) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A null value for *argument* yields undefined results.

%% Print a **%**; no argument is converted.

EXAMPLES

The command

```
example% printf '%s %s %s\n' Good Morning World
```

results in the output:

```
Good Morning World
```

The following command produces the same output.

```
example% printf '%2$s %s %1$s\n' World Good Morning
```

Here is an example that prints the first 6 characters of **$PATH** left-adjusted in a 10-character field:

```
example% printf 'First 6 chars of %s are %-10.6s.\n' $PATH $PATH
```

If **$PATH** has the value **/usr/bin:/usr/local/bin**, then the above command would print the following output:

```
First 6 chars of /usr/bin:/usr/local/bin are /usr/b .
```

SEE ALSO

printf(3S)

NAME

priocntl – display or set scheduling parameters of specified process(es)

SYNOPSIS

priocntl –l
priocntl –d [**–i** *idtype*] [*idlist*]
priocntl –s [**–c** *class*] [*class-specific options*] [**–i** *idtype*] [*idlist*]
priocntl –e [**–c** *class*] [*class-specific options*] *command* [*argument(s)*]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **priocntl** command displays or sets scheduling parameters of the specified process(es). It can also be used to display the current configuration information for the system's process scheduler or execute a command with specified scheduling parameters.

Processes fall into distinct classes with a separate scheduling policy applied to each class. The process classes currently supported are the real-time class, time-sharing class, and the interactive class. The characteristics of these classes and the class-specific options they accept are described below in the **USAGE** section under the headings **Real-Time Class**, **Time-Sharing Class**, and **Inter-Active Class**. With appropriate permissions, the **priocntl** command can change the class and other scheduling parameters associated with a running process.

In the default configuration, a runnable real-time process runs before any other process. Therefore, inappropriate use of real-time processes can have a dramatic negative impact on system performance.

If an *idlist* is present it must appear last on the command line and the elements of the list must be separated by white space. If no *idlist* is present an *idtype* argument of **pid**, **ppid**, **pgid**, **sid**, **class**, **uid**, or **gid** specifies the process ID, parent process ID, process group ID, session ID, class, user ID, or group ID respectively of the **priocntl** command itself.

The command

priocntl –d [**–i** *idtype*] [*idlist*]

displays the class and class-specific scheduling parameters of the process(es) specified by *idtype* and *idlist*.

The command

priocntl –s [**–c** *class*] [*class-specific options*] [**–i** *idtype*] [*idlist*]

sets the class and class-specific parameters of the specified processes to the values given on the command line. The **–c** *class* option specifies the class to be set. (The valid *class* arguments are **RT** for real-time **TS** for time-sharing or **IA** for inter-active.)

The class-specific parameters to be set are specified by the class-specific options as explained under the appropriate heading below. If the **–c** *class* option is omitted, *idtype* and *idlist* must specify a set of processes which are all in the same class, otherwise an error results. If no class-specific options are specified the process's class-specific parameters are set to the default values for the class specified by **–c** *class* (or to the default parameter values for the process's current class if the **–c** *class* option is also omitted).

In order to change the scheduling parameters of a process using **priocntl** the real or effective user ID of the user invoking **priocntl** must match the real or effective user ID of the receiving process or the effective user ID of the user must be super-user. These are the minimum permission requirements enforced for all classes. An individual class may impose additional permissions requirements when setting processes to that class or when setting class-specific scheduling parameters.

When *idtype* and *idlist* specify a set of processes, **priocntl** acts on the processes in the set in an implementation-specific order. If **priocntl** encounters an error for one or more of the target processes, it may or may not continue through the set of processes, depending on the nature of the error.

If the error is related to permissions, **priocntl** prints an error message and then continue through the process set, resetting the parameters for all target processes for which the user has appropriate permissions. If **priocntl** encounters an error other than permissions, it does not continue through the process set but prints an error message and exits immediately.

A special **sys** scheduling class exists for the purpose of scheduling the execution of certain special system processes (such as the swapper process). It is not possible to change the class of any process to **sys**. In addition, any processes in the **sys** class that are included in the set of processes specified by *idtype* and *idlist* are disregarded by **priocntl**. For example, if *idtype* were **uid**, an *idlist* consisting of a zero would specify all processes with a UID of zero except processes in the **sys** class and (if changing the parameters using the **–s** option) the **init** process.

The **init** process (process ID 1) is a special case. In order for the **priocntl** command to change the class or other scheduling parameters of the **init** process, *idtype* must be **pid** and *idlist* must be consist of only a 1. The **init** process may be assigned to any class configured on the system, but the time-sharing class is almost always the appropriate choice. (Other choices may be highly undesirable; see the *File System Administration* for more information.)

The command

priocntl –e [–c *class***]** [*class-specific options*] *command* [*argument* . . .]

executes the specified command with the class and scheduling parameters specified on the command line (*arguments* are the arguments to the command). If the **–c** *class* option is omitted the command is run in the user's current class.

OPTIONS

–l Display a list of the classes currently configured in the system along with class-specific information about each class. The format of the class-specific information displayed is described under USAGE.

–d Display the scheduling parameters associated with a set of processes.

–s Set the scheduling parameters associated with a set of processes.

–e Execute a specified command with the class and scheduling parameters associated with a set of processes.

–i *idtype* This option together with the *idlist* arguments (if any), specify one or more processes to which the **priocntl** command is to apply. The interpretation of *idlist* depends on the value of *idtype*. The valid *idtype* arguments and corresponding interpretations of *idlist* are as follows:

–i pid *idlist* is a list of process IDs. The **priocntl** command applies to the specified processes.

–i ppid *idlist* is a list of parent process IDs. The **priocntl** command applies to all processes whose parent process ID is in the list.

–i pgid *idlist* is a list of process group IDs. The **priocntl** command applies to all processes in the specified process groups.

–i sid *idlist* is a list of session IDs. The **priocntl** command applies to all processes in the specified sessions.

–i class *idlist* consists of a single class name (**RT** for real-time or **TS** for time-sharing or **IA** for inter-active). The **priocntl** command applies to all processes in the specified class.

–i uid *idlist* is a list of user IDs. The **priocntl** command applies to all processes with an effective user ID equal to an ID from the list.

–i gid *idlist* is a list of group IDs. The **priocntl** command applies to all processes with an effective group ID equal to an ID from the list.

–i all The **priocntl** command applies to all existing processes. No *idlist* should be specified (if one is it is ignored). The permission restrictions described below still apply.

If the **–i** *idtype* option is omitted when using the **–d** or **–s** options the default *idtype* of **pid** is assumed.

–c *class* Specifies the *class* to be set. (The valid *class* arguments are **RT** for real-time or **TS** for time-sharing or **IA** for inter-active.) If the specified class is not already configured, it will automatically be configured.

The valid class-specific options for setting real-time parameters are:

–p *rtpri* Set the real-time priority of the specified process(es) to *rtpri*.

–t *tqntm* [**–r** *res*]
Set the time quantum of the specified process(es) to *tqntm*. You may optionally specify a resolution as explained below.

The valid class-specific options for setting time-sharing parameters are:

–m *tsuprilim*
Set the user priority limit of the specified process(es) to *tsuprilim*.

–p *tsupri* Set the user priority of the specified process(es) to *tsupri*.

The valid class-specific options for setting inter-active parameters are:

–m *iamode* Mark the specified process(es) as currently interactive, or not. *iamode*.

USAGE

Real-Time Class

The real-time class provides a fixed priority preemptive scheduling policy for those processes requiring fast and deterministic response and absolute user / application control of scheduling priorities. If the real-time class is configured in the system it should have exclusive control of the highest range of scheduling priorities on the system. This ensures that a runnable real-time process is given CPU service before any process belonging to any other class.

The real-time class has a range of real-time priority (*rtpri*) values that may be assigned to processes within the class. Real-time priorities range from 0 to *x*, where the value of *x* is configurable and can be displayed for a specific installation that has already configured a real-time scheduler, by using the command

```
priocntl –l
```

The real-time scheduling policy is a fixed priority policy. The scheduling priority of a real-time process never changes except as the result of an explicit request by the user / application to change the *rtpri* value of the process.

For processes in the real-time class, the *rtpri* value is, for all practical purposes, equivalent to the scheduling priority of the process. The *rtpri* value completely determines the scheduling priority of a real-time process relative to other processes within its class. Numerically higher *rtpri* values represent higher priorities. Since the real-time class controls the highest range of scheduling priorities in the system it is guaranteed that the runnable real-time process with the highest *rtpri* value is always selected to run before any other process in the system.

In addition to providing control over priority, **priocntl** provides for control over the length of the time quantum allotted to processes in the real-time class. The time quantum value specifies the maximum amount of time a process may run assuming that it does not complete or enter a resource or event wait state (**sleep**). Note: If another process becomes runnable at a higher priority the currently running process may be preempted before receiving its full time quantum.

The command

```
priocntl –d [–i idtype] [idlist]
```

displays the real-time priority and time quantum (in millisecond resolution) for each real-time process in the set specified by *idtype* and *idlist*.

Any combination of the **–p** and **–t** options may be used with **priocntl –s** or **priocntl –e** for the real-time class. If an option is omitted and the process is currently real-time the associated parameter is unaffected. If an option is omitted when changing the class of a process to real-time from some other class, the associated parameter is set to a default value. The default value for *rtpri* is 0 and the default for time quantum is dependent on the value of *rtpri* and on the system configuration; see **rt_dptbl**(4).

When using the **–t** *tqntm* option you may optionally specify a resolution using the **–r** *res* option. (If no resolution is specified, millisecond resolution is assumed.) If *res* is specified it must be a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds. For example, specifying **–t 10 –r 100** would set the resolution to hundredths of a second and the resulting time quantum length would be 10/100 seconds (one tenth of a second). Although very fine (nanosecond) resolution may be specified, the time quantum length is rounded up by the system to the next integral multiple of the system clock's resolution. Requests for time quantums of zero or quantums greater than the (typically very large) implementation-specific maximum quantum result in an error.

In order to change the class of a process to real-time (from any other class) the user invoking **priocntl** must have super-user privilege. In order to change the *rtpri* value or time quantum of a real-time process the user invoking **priocntl** must either be super-user, or must currently be in the real-time class (shell running as a real-time process) with a real or effective user ID matching the real or effective user ID of the target process.

The real-time priority and time quantum are inherited across the **fork**(2) and **exec**(2) system calls.

Time-Sharing Class

The time-sharing scheduling policy provides for a fair and effective allocation of the CPU resource among processes with varying CPU consumption characteristics. The objectives of the time-sharing policy are to provide good response time to interactive processes and good throughput to CPU-bound jobs while providing a degree of user/application control over scheduling.

The time-sharing class has a range of time-sharing user priority (*tsupri*) values that may be assigned to processes within the class. User priorities range from *–x* to *+x*, where the value of *x* is configurable. The range for a specific installation can be displayed by using the command

```
priocntl –l
```

The purpose of the user priority is to provide some degree of user/application control over the scheduling of processes in the time-sharing class. Raising or lowering the *tsupri* value of a process in the time-sharing class raises or lowers the scheduling priority of the process. It is not guaranteed, however, that a time-sharing process with a higher *tsupri* value will run before one with a lower *tsupri* value. This is because the *tsupri* value is just one factor used to determine the scheduling priority of a time-sharing process. The system may dynamically adjust the internal scheduling priority of a time-sharing process based on other factors such as recent CPU usage.

In addition to the system-wide limits on user priority (displayed with **priocntl –l**), there is a per process user priority limit (*tsuprilim*), which specifies the maximum *tsupri* value that may be set for a given process.

The command

```
priocntl –d [–i idtype] [idlist]
```

displays the user priority and user priority limit for each time-sharing process in the set specified by *idtype* and *idlist*.

Any time-sharing process may lower its own *tsuprilim* (or that of another process with the same user ID). Only a time-sharing process with super-user privilege may raise a *tsuprilim*. When changing the class of a process to time-sharing from some other class, super-user privilege is required in order to set the initial *tsuprilim* to a value greater than zero.

Any time-sharing process may set its own *tsupri* (or that of another process with the same user ID) to any value less than or equal to the process's *tsuprilim*. Attempts to set the *tsupri* above the *tsuprilim* (and / or set the *tsuprilim* below the *tsupri*) result in the *tsupri* being set equal to the *tsuprilim*.

Any combination of the **−m** and **−p** options may be used with **priocntl −s** or **priocntl −e** for the time-sharing class. If an option is omitted and the process is currently time-sharing the associated parameter is normally unaffected. The exception is when the **−p** option is omitted and **−m** is used to set a *tsuprilim* below the current *tsupri*. In this case the *tsupri* is set equal to the *tsuprilim* which is being set. If an option is omitted when changing the class of a process to time-sharing from some other class, the associated parameter is set to a default value. The default value for *tsuprilim* is 0 and the default for *tsupri* is to set it equal to the *tsuprilim* value which is being set.

The time-sharing user priority and user priority limit are inherited across the **fork**(2) and **exec**(2) system calls.

Inter-Active Class

The inter-active scheduling policy provides for a fair and effective allocation of the CPU resource among processes with varying CPU consumption characteristics while providing good responsiveness for user interaction. The objectives of the inter-active policy are to provide good response time to interactive processes and good throughput to CPU-bound jobs. Only the super user has access to the inter-active class, the user has no control over scheduling policys.

EXAMPLES

Real-Time Class examples follow:

```
example% priocntl −s −c RT −t 1 −r 10 −i idtype idlist
```

The above example sets the class of any non-real-time processes selected by *idtype* and *idlist* to real-time and sets their real-time priority to the default value of 0. The real-time priorities of any processes currently in the real-time class are unaffected. The time quantums of all of the specified processes are set to 1 / 10 seconds.

```
example% priocntl −e −c RT −p 15 −t 20 command
```

This example executes *command* in the real-time class with a real-time priority of 15 and a time quantum of 20 milliseconds.

Time-Sharing Class examples follow:

```
example% priocntl −s −c TS −i idtype idlist
```

The above example sets the class of any non-time-sharing processes selected by *idtype* and *idlist* to time-sharing and sets both their user priority limit and user priority to 0. Processes already in the time-sharing class are unaffected.

This example executes *command* with the arguments *arguments* in the time-sharing class with a user priority limit of 0 and a user priority of −15.

```
example% priocntl −e −c TS −m 0 −p −15 command [arguments]
```

SEE ALSO

nice(1), **ps**(1), **exec**(2), **fork**(2), **priocntl**(2), **rt_dptbl**(4)

DIAGNOSTICS

priocntl prints the following error messages:

Process(es) not found
None of the specified processes exists.

Specified processes from different classes
The −**s** option is being used to set parameters, the −**c** *class* option is not present, and processes from more than one class are specified.

Invalid option or argument
An unrecognized or invalid option or option argument is used.

NAME

prof – display profile data

SYNOPSIS

prof [–**a** | **c** | **n** | **t**] [–**o** | **x**] [–**g** | **l**] [–**C**] [–**h**] [–**m** *mdata*] [–**s**] [–**V** *prog*] [–**z**]

DESCRIPTION

The **prof** command interprets a profile file produced by the **monitor** function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

OPTIONS

The mutually exclusive options –**a**, –**c**, –**n**, and –**t** determine the type of sorting of the output lines:

–**a** Sort by increasing symbol address.

–**c** Sort by decreasing number of calls.

–**n** Sort lexically by symbol name.

–**t** Sort by decreasing percentage of total time (default).

The mutually exclusive options –**o** and –**x** specify the printing of the address of each symbol monitored:

–**o** Print each symbol address (in octal) along with the symbol name.

–**x** Print each symbol address (in hexadecimal) along with the symbol name.

The mutually exclusive options –**g** and –**l** control the type of symbols to be reported. The –**l** option must be used with care; it applies the time spent in a static function to the preceding (in memory) global function, instead of giving the static function a separate entry in the report. If all static functions are properly located (see example below), this feature can be very useful. If not, the resulting report may be misleading.

Assume that **A** and **B** are global functions and only **A** calls static function **S**. If **S** is located immediately after A in the source code (that is, if **S** is properly located), then, with the –**l** option, the amount of time spent in **A** can easily be determined, including the time spent in **S**. If, however, both **A** and **B** call **S**, then, if the –**l** option is used, the report will be misleading; the time spent during **B**’s call to **S** will be attributed to **A**, making it appear as if more time had been spent in **A** than really had. In this case, function **S** cannot be properly located.

–**g** Include static (non-global) functions.

–**l** Do not include static (non-global) functions (default).

The following options may be used in any combination:

–**C** Demangle C++ symbol names before printing them out.

–**h** Suppress the heading normally printed on the report. This is useful if the report is to be processed further.

–m *mdata*
Use file *mdata* instead of **mon.out** as the input profile file.

–s	Print a summary of several of the monitoring parameters and statistics on the standard error output.
–V	Print **prof** version information on the standard error output.
–z	Include all symbols in the profile range, even if associated with zero number of calls and zero time.

A program creates a profile file if it has been link edited with the **–p** option of **cc**(1B). This option to the **cc**(1B) command arranges for calls to **monitor** at the beginning and end of execution. It is the call to **monitor** at the end of execution that causes the system to write a profile file. The number of calls to a function is tallied if the **–p** option was used when the file containing the function was compiled.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(5)).

ENVIRONMENT

PROFDIR The name of the file created by a profiled program is controlled by the environment variable **PROFDIR**. If **PROFDIR** is not set, **mon.out** is produced in the directory current when the program terminates. If **PROFDIR=***string*, *string*/*pid***.***progname* is produced, where *progname* consists of **argv[0]** with any path prefix removed, and *pid* is the process ID of the program. If **PROFDIR** is set, but null, no profiling output is produced.

FILES

mon.out	default profile file
a.out	default namelist (object) file

SEE ALSO

cc(1B), **exit**(2), **profil**(2), **malloc**(3C), **monitor**(3C), **malloc**(3X), **prof**(5)

The **lprof** section in *Programming Utilities Guide*

NOTES

The times reported in successive identical runs may show variances because of varying cache-hit ratios that result from sharing the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may ‘‘beat’’ with loops in a program, grossly distorting measurements. Call counts are always recorded precisely, however.

Only programs that call **exit** or return from **main** are guaranteed to produce a profile file, unless a final call to **monitor** is explicitly coded.

The times for static functions are attributed to the preceding external text symbol if the **–g** option is not used. However, the call counts for the preceding function are still correct; that is, the static function call counts are not added to the call counts of the external function.

If more than one of the options **–t**, **–c**, **–a**, and **–n** is specified, the last option specified is used and the user is warned.

Profiling may be used with dynamically linked executables, but care must be applied. Currently, shared objects cannot be profiled with **prof**. Thus, when a profiled, dynamically linked program is executed, only the ‘‘main’’ portion of the image is sampled. This means that all time spent outside of the ‘‘main’’ object, that is, time spent in a shared object, will not be included in the profile summary; the total time reported for the program may be less than the total time used by the program.

Because the time spent in a shared object cannot be accounted for, the use of shared objects should be minimized whenever a program is profiled with **prof**. If desired, the program should be linked to the profiled version of a library (or to the standard archive version if no profiling version is available), instead of the shared object to get profile information on the functions of a library. Versions of profiled libraries may be supplied with the system in the **/usr/lib/libp** directory. Refer to compiler driver documentation on profiling.

Consider an extreme case. A profiled program dynamically linked with the shared C library spends 100 units of time in some **libc** routine, say, **malloc( )**. Suppose **malloc( )** is called only from routine **B and B** consumes only 1 unit of time. Suppose further that routine **A** consumes 10 units of time, more than any other routine in the ‘‘main’’ (profiled) portion of the image. In this case, **prof** will conclude that most of the time is being spent in **A** and almost no time is being spent in **B**. From this it will be almost impossible to tell that the greatest improvement can be made by looking at routine **B** and not routine **A**. The value of the profiler in this case is severely degraded; the solution is to use archives as much as possible for profiling.

NAME

ps – report process status

SYNOPSIS

ps [–**acdefjl**] [–**g** *grplist*] [–**p** *proclist*] [–**s** *sidlist*] [–**t** *term*] [–**u** *uidlist*]

AVAILABILITY

SUNWcsu

DESCRIPTION

ps prints information about active processes. Without *options*, **ps** prints information about processes associated with the controlling terminal. The output contains only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the *options*.

Some options accept lists as arguments. Items in a list can be either separated by commas or else enclosed in double quotes and separated by commas or spaces. Values for *proclist* and *grplist* must be numeric.

OPTIONS

–**a**	Print information about **a**ll processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
–**c**	Print information in a format that reflects scheduler properties as described in **priocntl**(1). The –**c** option affects the output of the –**f** and –**l** options, as described below.
–**d**	Print information about all processes except session leaders.
–**e**	Print information about **e**very process now running.
–**f**	Generate a **f**ull listing. (See below for significance of columns in a full listing.)
–**j**	Print session ID and process group ID.
–**l**	Generate a **l**ong listing. (See below.)
–**g** *grplist*	List only process data whose group leader's ID number(s) appears in *grplist*. (A group leader is a process whose process ID number is identical to its process group ID number.
–**p** *proclist*	List only process data whose process ID numbers are given in *proclist*.
–**s** *sidlist*	List information on all session leaders whose IDs appear in *sidlist*.
–**t** *term*	List only process data associated with *term*. Terminal identifiers are specified as a device file name, and an identifier. For example, **term/a**, or **pts/0**.
–**u** *uidlist*	List only process data whose user ID number or login name is given in *uidlist*. In the listing, the numerical user ID will be printed unless you give the –**f** option, which prints the login name.

DISPLAY FORMATS

Under the –**f** option, **ps** tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the –**f** option, in square brackets.

The column headings and the meaning of the columns in a **ps** listing are given below; the letters **f** and **l** indicate the option (**f**ull or **l**ong, respectively) that causes the corresponding heading to appear; **all** means that the heading always appears. Note: These two options determine only what information is provided for a process; they do not determine which processes will be listed.

F	(l)	Flags (hexadecimal and additive) associated with the process. These flags are available for historical purposes; no meaning should be currently ascribed to them.
S	(l)	The state of the process: O Process is running on a processor. S Sleeping: process is waiting for an event to complete. R Runnable: process is on run queue. I Idle: process is being created. Z Zombie state: process terminated and parent not waiting. T Traced: process stopped by a signal because parent is tracing it. X SXBRK state: process is waiting for more primary memory.
UID	(f,l)	The user ID number of the process owner (the login name is printed under the –**f** option).
PID	(all)	The process ID of the process (this datum is necessary in order to kill a process).
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling. Not printed when the –**c** option is used.
CLS	(f,l)	Scheduling class. Printed only when the –**c** option is used.
PRI	(l)	The priority of the process. Without the –**c** option, higher numbers mean lower priority. With the –**c** option, higher numbers mean higher priority.
NI	(l)	Nice value, used in priority computation. Not printed when the –**c** option is used. Only processes in the time-sharing class have a nice value.
ADDR	(l)	The memory address of the process.
SZ	(l)	The size (in pages or clicks) of the swappable process's image in main memory.

WCHAN	(l)	The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).
STIME	(f)	The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the **ps** inquiry is executed is given in months and days.)
TTY	(all)	The controlling terminal for the process (the message, **?**, is printed when there is no controlling terminal).
TIME	(all)	The cumulative execution time for the process.
COMMAND	(all)	The command name (the full command name and its arguments are printed under the **–f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **ps** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **ps** behaves.

LC_CTYPE

Determines how **ps** handles characters. When **LC_CTYPE** is set to a valid value, **ps** can display and handle text and filenames containing valid characters for that locale. **ps** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **ps** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME

Determines how **ps** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/dev **/dev/sxt/*** **/dev/tty*** **/dev/xt/***	terminal (‘‘tty’’) names searcher files
/dev/kmem	kernel virtual memory
/dev/mem	memory
/dev/swap	the default swap device
/etc/passwd	UID information supplier

/tmp/ps_data	internal data structure

SEE ALSO

kill(1), **nice**(1), **priocntl**(1), **getty**(1M), **environ**(5)

NOTES

Things can change while **ps** is running; the snap-shot it gives is true only for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, **ps** checks **stdin**, **stdout**, and **stderr** in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if **stdin**, **stdout**, and **stderr** are all redirected, **ps** will not find a controlling terminal, so there will be no report.

On a heavily loaded system, **ps** may report an **lseek** error and exit. **ps** may seek to an invalid user area address: having obtained the address of a process' user area, **ps** may not be able to seek to that address before the process exits and the address becomes invalid.

ps −ef may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

NAME

ps – display the status of current processes

SYNOPSIS

/usr/ucb/ps [**–acglnrSuUvwx**] [**–t** *term*] [*num*]

AVAILABILITY

SUNWscpu

DESCRIPTION

The **ps** command displays information about processes. Normally, only those processes that are running with your effective user ID and are attached to a controlling terminal (see **termio**(7)) are shown. Additional categories of processes can be added to the display using various options. In particular, the **–a** option allows you to include processes that are not owned by you (that do not have your user ID), and the **–x** option allows you to include processes without control terminals. When you specify both **–a** and **–x ,** you get processes owned by anyone, with or without a control terminal. The **–r** option restricts the list of processes printed to running and runnable processes.

ps displays the process ID, under PID; the control terminal (if any), under TT; the cpu time used by the process so far, including both user and system time, under TIME; the state of the process, under S; and finally, an indication of the COMMAND that is running.

The state is given by a single letter from the following:

O	Process is running on a processor.
S	Sleeping. Process is waiting for an event to complete.
R	Runnable. Process is on run queue.
I	Idle. Process is being created.
Z	Zombie state. Process terminated and parent not waiting.
T	Traced. Process stopped by a signal because parent is tracing it.
X	**SXBRK** state. Process is waiting for more primary memory.

OPTIONS

The following options must all be combined to form the first argument:

–a	Include information about processes owned by others.
–c	Display the command name, as stored internally in the system for purposes of accounting, rather than the command arguments, which are kept in the process' address space. This is more reliable, if less informative, since the process is free to destroy the latter information.
–g	Display all processes. Without this option, **ps** only prints interesting processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to login on free terminals.
–l	Display a long listing, with fields **F , PPID , CP , PRI , NI , SZ , RSS** and **WCHAN** as described below.
–n	Produce numerical output for some fields. In a user listing, the **USER** field is replaced by a **UID** field.
–r	Restrict output to running and runnable processes.

–S	Display accumulated CPU time used by this process and all of its reaped children.
–u	Display user-oriented output. This includes fields **USER , SZ , RSS** and **START** as described below.
–U	Update a private database where **ps** keeps system information.
–v	Display a version of the output containing virtual memory. This includes fields **SIZE** and **RSS ,** described below.
–w	Use a wide output format (132 columns rather than 80); if repeated, that is, **–ww ,** use arbitrarily wide output. This information is used to decide how much of long commands to print.
–x	Include processes with no controlling terminal.
–t *term*	List only process data associated with the terminal, *term*. Terminal identifiers may be specified in one of two forms: the device's file name (for example, **tty04** or **term/14**) or, if the device's file name starts with **tty**, just the digit identifier (for example, **04**).
num	A process number may be given, in which case the output is restricted to that process. This option must be supplied last.

DISPLAY FORMATS

Fields that are not common to all output formats:

USER	Name of the owner of the process.
NI	Process scheduling increment (see **getpriority**(3B) and **nice**(3B)).
SIZE **SZ**	The combined size of the data and stack segments (in kilobyte units)
RSS	Real memory (resident set) size of the process (in kilobyte units).
UID	Numerical user-ID of process owner.
PPID	Numerical ID of parent of process.
CP	Short-term CPU utilization factor (used in scheduling).
PRI	The priority of the process (higher numbers mean lower priority).
START	The starting time of the process, given in hours, minutes, and seconds. A process begun more than 24 hours before the **ps** inquiry is executed is given in months and days.
WCHAN	The address of an event for which the process is sleeping, or in **SXBRK** state (if blank, the process is running).
F	Flags (hexadecimal and additive) associated with the process: **00** Process has terminated. Process table now available. **01** A system process, always in primary memory. **02** Parent is tracing process. **04** Tracing parent's signal has stopped process. Parent is waiting, see **ptrace**(2). **08** Process is currently in primary memory.

10	Process currently in primary memory, locked until an event is completed.

A process that has exited and has a parent, but has not yet been waited for by the parent is marked < **defunct >;** otherwise, **ps** tries to determine the command name and arguments given when the process was created by examining the user block.

FILES

/dev	
/dev/kmem	kernel virtual memory
/dev/mem	memory
/dev/swap	default swap device
/dev/sxt/∗	
/dev/tty∗	
/dev/xt/∗	terminal (**tty**) names searcher files
/etc/passwd	UID information supplier
/etc/ps_data	internal data structure

SEE ALSO

kill(1), **whodo**(1M), **lseek**(2), **getpriority**(3B), **nice**(3B)

NOTES

Things can change while **ps** is running; the picture it gives is only a close approximation to the current state. Some data printed for defunct processes is irrelevant.

If no *term* or *num* is specified, **ps** checks the standard input, the standard output, and the standard error in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if the standard input, the standard output, and the standard error are all redirected, **ps** will not find a controlling terminal, so there will be no report.

On a heavily loaded system, **ps** may report an **lseek**(2) error and exit. **ps** may seek to an invalid user area address, having obtained the address of process' user area, **ps** may not be able to seek to that address before the process exits and the address becomes invalid.

NAME

pwd – working directory name

SYNOPSIS

pwd

AVAILABILITY

SUNWcsu

DESCRIPTION

pwd prints the path name of the working (current) directory.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **pwd** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **pwd** behaves.

LC_CTYPE
Determines how **pwd** handles characters. When **LC_CTYPE** is set to a valid value, **pwd** can display and handle text and filenames containing valid characters for that locale. **pwd** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **pwd** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

cd(1), **shell_builtins**(1), **environ**(5)

DIAGNOSTICS

‘‘**Cannot open ..**’’ and ‘‘**Read error in ..**’’ indicate possible file system trouble and should be referred to a UNIX system administrator.

NOTES

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd**(1) command with a full path name to correct this situation.

NAME

rcp – remote file copy

SYNOPSIS

rcp [−**p**] *filename1 filename2*
rcp [−**pr**] *filename*. . .*directory*

AVAILABILITY

SUNWcsu

DESCRIPTION

The **rcp** command copies files between machines. Each *filename* or *directory* argument is either a remote file name of the form:

*hostname***:***path*

or a local file name (containing no **:** characters, or a **/** before any **:** characters).

If a *filename* is not a full path name, it is interpreted relative to your home directory on *hostname*. A *path* on a remote host may be quoted (using \ , **"** , or **’**) so that the metacharacters are interpreted remotely.

rcp does not prompt for passwords; your current local user name must exist on *hostname* and allow remote command execution by **rsh**(1).

rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form

*username***@***hostname***:***filename*

to use *username* rather than your current local user name as the user name on the remote host. **rcp** also supports Internet domain addressing of the remote host, so that:

*username***@***host***.***domain***:***filename*

specifies the username to be used, the hostname, and the domain in which that host resides. Filenames that are not full path names will be interpreted relative to the home directory of the user named *username*, on the remote host.

OPTIONS

−**p**	Attempt to give each copy the same modification times, access times, and modes as the original file.
−**r**	Copy each subtree rooted at *filename*; in this case the destination must be a directory.

FILES

$HOME/.profile

SEE ALSO

cpio(1), **ftp**(1), **rlogin**(1), **rsh**(1), **tar**(1), **hosts.equiv**(4)

NOTES

rcp is meant to copy between different hosts; attempting to **rcp** a file onto itself, as with:

```
rcp tmp/file myhost:/tmp/file
```

results in a severely corrupted file.

rcp may not correctly fail when the target of a copy is a file instead of a directory.

rcp can become confused by output generated by commands in a **$HOME/.profile** on the remote host.

rcp requires that the source host have permission to execute commands on the remote host when doing third-party copies.

rcp does not properly handle symbolic links. Use **tar** (see **tar**(1)) or **cpio** (see **cpio**(1)) piped to **rsh** to obtain remote copies of directories containing symbolic links or named pipes.

If you forget to quote metacharacters intended for the remote host you get an incomprehensible error message.

NAME

rdist – remote file distribution program

SYNOPSIS

rdist [–**b**] [**-D**] [–**h**] [–**i**] [–**n**] [–**q**] [–**R**] [–**v**] [–**w**] [–**y**]
[–**d** *macro* = *value*] [–**f** *distfile*] [–**m** *host*] . . . [*package* . . .]

rdist [–**b**] [–**D**] [–**h**] [–**i**] [–**n**] [–**q**] [–**R**] [–**v**] [–**w**] [–**y**]
–**c** *pathname* . . . [*login*@] *hostname* [**:***destpath*]

AVAILABILITY

SUNWcsu

DESCRIPTION

rdist maintains copies of files on multiple hosts. It preserves the owner, group, mode, and modification time of the master copies, and can update programs that are executing. Normally, a copy on a remote host is updated if its size or modification time differs from the original on the local host. **rdist** reads the indicated *distfile* for instructions on updating files and / or directories. If *distfile* is '–', the standard input is used. If no –**f** option is present, **rdist** first looks in its working directory for **distfile**, and then for **Distfile**, for instructions.

rdist updates each *package* specified on the command line; if none are given, all packages are updated according to their entries in the *distfile*.

In order to be able to use **rdist** across machines, each host machine must have a **/etc/host.equiv** file, or the user must have an entry in the **.rhosts** file in the home directory. See **hosts.equiv**(4) for more information.

OPTIONS

–**b**	Binary comparison. Perform a binary comparison and update files if they differ, rather than merely comparing dates and sizes.
–**D**	Enable debugging.
–**h**	Follow symbolic links. Copy the file that the link points to rather than the link itself.
–**i**	Ignore unresolved links. **rdist** will normally try to maintain the link structure of files being transferred and warn the user if all the links cannot be found.
–**n**	Print the commands without executing them. This option is useful for debugging a distfile.
–**q**	Quiet mode. Do not display the files being updated on the standard output.
–**R**	Remove extraneous files. If a directory is being updated, remove files on the remote host that do not correspond to those in the master (local) directory. This is useful for maintaining truly identical copies of directories.
–**v**	Verify that the files are up to date on all the hosts. Any files that are out of date are displayed, but no files are updated, nor is any mail sent.
–**w**	Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when

renaming files. This preserves the directory structure of the files being copied, instead of flattening the directory structure. For instance, renaming a list of files such as **dir1/dir2** to **dir3** would create files **dir3/dir1** and **dir3/dir2** instead of **dir3** and **dir3**. When the −**w** option is used with a filename that begins with **~,** everything except the home directory is appended to the destination name.

−**y**	Younger mode. Do not update remote copies that are younger than the master copy, but issue a warning message instead.
−**d** *macro*=*value*	Define *macro* to have *value*. This option is used to define or override macro definitions in the distfile. *value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by white space.
−**c** *pathname* . . . [*login*@]*hostname*[:*destpath*]	Update each *pathname* on the named host. (Relative filenames are taken as relative to your home directory.) If the '*login*@' prefix is given, the update is performed with the user ID of *login*. If the ':*destpath*' is given, the remote file is installed as that pathname.
−**f** *distfile*	Use the description file *distfile*. A '−' as the *distfile* argument denotes the standard input.
−**m** *host*	Limit which machines are to be updated. Multiple −**m** arguments can be given to limit updates to a subset of the hosts listed in the distfile.

USAGE

Packages

A typical package begins with a label composed of the package name followed by a colon:

package:

This label allows you to group any number of file-to-host and file-to-timestamp mappings into a single distribution package. If no package label appears in the distfile, the default package includes all mappings in the file.

A file-to-host mapping specifies a list of files or directories to distribute, their destination host(s), and any **rdist** primitives to use in performing the update. A mapping of this sort takes the form:

(*pathname* . . . **)** → **(** *hostname* . . . **)** *primitive* ; [*primitive* ;] . . .

In this case, each *pathname* is the full pathname of a local file or directory to distribute; each *hostname* is the name of a remote host on which those files are to be copied, and *primitive* is one of the **rdist** primitive listed under *Primitives*, below. A *hostname* can also take the form:

login@*hostname*

in which case the update is performed as the user named *login*.

A file-to-timestamp mapping is used to monitor which local files are updated with respect to a local "timestamp" file. This mapping takes the form:

(*filename* . . . **)** **::** *timestamp-file primitive* **;** [*primitive* **;**] . . .

In this case, *timestamp-file* is the name of a file, the modification time of which is compared with each named file on the local host. If a file is newer than *time-stamp-file*, **rdist** displays a message to that effect.

White Space Characters

NEWLINE, TAB, and SPACE characters are all treated as white space; a mapping continues across input lines until the start of the next mapping: either a single *filename* followed by a '→' or the opening parenthesis of a filename list.

Comments

Comments begin with **#** and end with a NEWLINE.

Macros

rdist has a limited macro facility. Macros are only expanded in filename or hostname lists, and in the argument lists of certain primitives. Macros cannot be used to stand for primitives or their options, or the **'→'** or **'::'** symbols.

A macro definition is a line of the form:

macro **=** *value*

A macro reference is a string of the form:

${*macro***}**

although (as with **make**(1S)) the braces can be omitted if the macro name consists of just one character.

Metacharacters

The shell meta-characters: **[**, **]**, **{**, **}**, ∗ and **?** are recognized and expanded (on the local host only) just as they are with **csh**(1). Metacharacters can be escaped by prepending a backslash.

The **˜** character is also expanded in the same way as with **csh**, however, it is expanded separately on the local and destination hosts.

Filenames

File names that do not begin with '**/**' or '**˜**' are taken to be relative to user's home directory on each destination host; they are *not* relative to the current working directory. Multiple file names must be enclosed within parentheses.

Primitives

The following primitives can be used to specify actions **rdist** is to take when updating remote copies of each file.

install [–**b**] [–**h**] [–**i**] [–**R**] [–**v**] [–**w**] [–**y**] [*newname*]

Copy out-of-date files and directories (recursively). If no **install** primitive appears in the package entry, or if no *newname* option is given, the name of the local file is given to the remote host's copy. If absent from the remote host, parent directories in a filename's path are created. To help prevent disasters, a non-empty directory on a target host is not replaced with a regular file or a symbolic link by **rdist**. However, when using the –**R** option, a non-empty directory is removed if the corresponding filename is completely absent on the master host.

The options for **install** have the same semantics as their command line counterparts, but are limited in scope to a particular map. The login name used on the destination host is the same as the local host unless the destination name is of the format *login@host*. In that case, the update is performed under the username *login*.

notify *address* . . .

Send mail to the indicated TCP / IP *address* of the form:

user@host

that lists the files updated and any errors that may have occurred. If an address does not contain a '*@host*' suffix, **rdist** uses the name of the destination host to complete the address.

except *filename* . . .

Omit from updates the files named as arguments.

except_pat *pattern* . . .

Omit from updates the filenames that match each regular-expression *pattern* (see **ed**(1) for more information on regular expressions). Note that **'\'** and **'$'** characters must be escaped in the distfile. Shell variables can also be used within a pattern, however shell filename expansion is not supported.

special [*filename*] . . . "*command-line*"

Specify a Bourne shell, **sh**(1) command line to execute on the remote host after each named file is updated. If no *filename* argument is present, the *command-line* is performed for every updated file, with the shell variable **FILE** set to the file's name on the local host. The quotation marks allow *command-line* to span input lines in the distfile; multiple shell commands must be separated by semicolons (;).

The default working directory for the shell executing each *command-line* is the user's home directory on the remote host.

EXAMPLES

The following sample distfile instructs **rdist** to maintain identical copies of a shared library, a shared-library initialized data file, several include files, and a directory, on hosts named **hermes** and **magus**. On **magus**, commands are executed as root. **rdist** notifies **merlin@druid** whenever it discovers that a local file has changed relative to a timestamp file.

```
HOSTS = ( hermes root@magus )

FILES = ( /usr/local/lib/libcant.so.1.1
        /usrlocal/lib/libcant.sa.1.1 /usr/local/include/{*.h}
        /usr/local/bin )

(${FILES}) → (${HOSTS})
        install −R ;
${FILES} :: /usr/local/lib/timestamp
        notify merlin@druid ;
```

FILES

˜/.rhosts	user's trusted hosts and users
/etc/host.equiv	system trusted hosts and users
/tmp/rdist*	temporary file for update lists

SEE ALSO

csh(1), **ed**(1), **make**(1S), **sh**(1), **stat**(2)

DIAGNOSTICS

A complaint about mismatch of **rdist** version numbers may really stem from some problem with starting your shell, for example, you are in too many groups.

WARNINGS

root does not have its accustomed access privileges on NFS mounted file systems. Using **rdist** to copy to such a file system may fail, or the copies may be owned by user "nobody."

BUGS

Source files must reside or be mounted on the local host.

There is no easy way to have a special command executed only once after all files in a directory have been updated.

Variable expansion only works for name lists; there should be a general macro facility.

rdist aborts on files that have a negative modification time (before Jan 1, 1970).

There should be a "force" option to allow replacement of non-empty directories by regular files or symlinks. A means of updating file modes and owners of otherwise identical files is also needed.

NAME

read – shell built-in function to receive from standard input (keyboard)

SYNOPSIS

sh

read *name* . . .

csh

set *variable* **= $<**

ksh

read [**–prsu**[*n*]] [*name***?***prompt*] [*name* . . .]

DESCRIPTION

sh

One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using **\newline**. Characters other than **newline** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is **0**, unless an EOF is encountered.

csh

The notation
set *variable* **= $<**
loads one line of standard input as the value for *variable.* (See **csh**(1)).

ksh

The shell input mechanism. One line is read and is broken up into fields using the characters in **IFS** as separators. The escape character, **(\)**, is used to remove any special meaning for the next character and for line continuation. In raw mode, **–r,** the \ character is not treated specially. The first field is assigned to the first *name*, the second field to the second *name*, etc., with leftover fields assigned to the last *name*. The **–p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **–s** flag is present, the input will be saved as a command in the history file. The flag **–u** can be used to specify a one digit file descriptor unit *n* to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is 0. If *name* is omitted then **REPLY** is used as the default *name*. The exit status is 0 unless the input file is not open for reading or an end-of-file is encountered. An end-of-file with the **–p** option causes cleanup for this process so that another can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* on standard error when the shell is interactive. The exit status is 0 unless an end-of-file is encountered.

SEE ALSO

csh(1), **ksh**(1), **sh**(1), **set**(1)

NAME

readfile, longline – reads file, gets longest line

SYNOPSIS

readfile *filename*

longline [*filename*]

DESCRIPTION

The **readfile** function reads *filename* and copies it to *stdout*. No translation of NEWLINE is done. It keeps track of the longest line it reads and if there is a subsequent call to **longline**, the length of that line, including the NEWLINE character, is returned.

The **longline** function returns the length, including the NEWLINE character, of the longest line in *filename*. If *filename* is not specified, it uses the file named in the last call to **readfile**.

EXAMPLES

Here is a typical use of **readfile** and **longline** in a text frame definition file:

```
  .
  .
  .
text="`readfile myfile`"
columns=`longline`
  .
  .
  .
```

SEE ALSO

cat(1)

DIAGNOSTICS

If *filename* does not exist, **readfile** will return FALSE (that is, the expression will have an error return).

longline returns 0 if a **readfile** has not previously been issued.

NOTES

More than one descriptor can call **readfile** in the same frame definition file. In text frames, if one of those calls is made from the **text** descriptor, then a subsequent use of **longline** will always get the longest line of the file read by the **readfile** associated with the **text** descriptor, even if it was not the most recent use of **readfile**.

NAME

readonly – shell built-in function to protect the value of the given variable from reassignment

SYNOPSIS

sh

readonly [*name* . . .]

ksh

†† **readonly** [*name*[**=***value*]] . . .

DESCRIPTION

sh

The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

ksh

The given *name*s are marked **readonly** and these names cannot be changed by subsequent assignment.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

ksh(1), **sh**(1), **typeset**(1)

NAME

refer – expand and insert references from a bibliographic database

SYNOPSIS

refer [–**ben**] [–**a***r*] [–**c***string*] [–**k***x*] [–**l***m,n*] [–**p** *filename*] [–**s***keys*] *filename*. . .

AVAILABILITY

SUNWdoc

DESCRIPTION

refer is a preprocessor for **nroff**(1), or **troff**(1), that finds and formats references. The input files (standard input by default) are copied to the standard output, except for lines between '**.[**' and '**.]**' command lines, Such lines are assumed to contain keywords as for **lookbib**(1), and are replaced by information from a bibliographic data base. The user can avoid the search, override fields from it, or add new fields. The reference data, from whatever source, is assigned to a set of **troff** strings. Macro packages such as **ms**(5) print the finished reference text from these strings. A flag is placed in the text at the point of reference. By default, the references are indicated by numbers.

When **refer** is used with **eqn**(1), **neqn**, or **tbl**(1), **refer** should be used first in the sequence, to minimize the volume of data passed through pipes.

OPTIONS

–b Bare mode — do not put any flags in text (neither numbers or labels).

–e Accumulate references instead of leaving the references where encountered, until a sequence of the form:

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

–n Do not search the default file.

–a*r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted, all author names are reversed.

–c*string*
Capitalize (with SMALL CAPS) the fields whose key-letters are in *string*.

–k*x* Instead of numbering references, use labels as specified in a reference data line beginning with the characters *%x*; By default, *x* is **L**.

–l*m,n* Instead of numbering references, use labels from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either of *m* or *n* is omitted, the entire name or date, respectively, is used.

–p *filename*
Take the next argument as a file of references to be searched. The default file is searched last.

–s*keys* Sort references by fields whose key-letters are in the *keys* string, and permute reference numbers in the text accordingly. Using this option implies the **–e** option. The key-letters in *keys* may be followed by a number indicating how many such fields are used, with a **+** sign taken as a very large number. The default is **AD**, which sorts on the senior author and date. To sort on all authors and then the date, for instance, use the options '**–sA+T**'.

FILES

/usr/lib/refer	directory of programs
/usr/lib/refer/papers	directory of default publication lists and indexes

SEE ALSO

addbib(1), **eqn**(1), **indxbib**(1), **lookbib**(1), **nroff**(1), **roffbib**(1), **sortbib**(1), **tbl**(1), **troff**(1)

NAME

regcmp – regular expression compile

SYNOPSIS

regcmp [–] *filename*. . .

DESCRIPTION

The **regcmp** command performs a function similar to **regcmp** and, in most cases, precludes the need for calling **regcmp** from C programs. Bypassing **regcmp** saves on both execution time and program size. The command **regcmp** compiles the regular expressions in *filename* and places the output in *filename*.**i**.

OPTIONS

– If the – option is used, the output is placed in *filename*.**c**. The format of entries in *filename* is a name (C variable) followed by one or more blanks followed by one or more regular expressions enclosed in double quotes. The output of **regcmp** is C source code. Compiled regular expressions are represented as **extern char** vectors. *filename*.**i** files may thus be **#include**d in C programs, or *filename*.**c** files may be compiled and later loaded. In the C program that uses the **regcmp** output, **regex(abc,line)** applies the regular expression named **abc** to **line**. Diagnostics are self-explanatory.

EXAMPLES

```
name   "([A–Za–z][A–Za–z0–9_]*)$0"
telno  " \({0,1}([2–9][01][1–9])$0\){0,1} *"
       "([2–9][0–9]{2})$1[ –]{0,1}"
       "([0–9]{4})$2"
```

The three arguments to **telno** shown above must all be entered on one line.

In the C program that uses the **regcmp** output,

```
regex(telno, line, area, exch, rest)
```

applies the regular expression named **telno** to **line**.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **regcmp** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **regcmp** behaves.

LC_CTYPE

Determines how **regcmp** handles characters. When **LC_CTYPE** is set to a valid value, **regcmp** can display and handle text and filenames containing valid characters for that locale. **regcmp** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **regcmp** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

regcmp(3G), **environ**(5)

NAME

regex – match patterns against a string

SYNOPSIS

regex [**–e**] [**–v** "*string*"] [*pattern template*] ... *pattern* [*template*]

DESCRIPTION

The **regex** command takes a string from *the standard input*, and a list of *pattern* / *template* pairs, and runs **regex ( )** to compare the string against each *pattern* until there is a match. When a match occurs, **regex** writes the corresponding *template* to *the standard output* and returns TRUE. The last (or only) *pattern* does not need a template. If that is the pattern that matches the string, the function simply returns TRUE. If no match is found, **regex** returns FALSE.

The argument *pattern* is a regular expression of the form described in **regex ( ).** In most cases *pattern* should be enclosed in single quotes to turn off special meanings of characters. Note that only the final *pattern* in the list may lack a *template* .

The argument *template* may contain the strings **$m0** through **$m9 ,** which will be expanded to the part of *pattern* enclosed in **(. . .)$0** through **(. . .)$9** constructs (see examples below). Note that if you use this feature, you must be sure to enclose *template* in single quotes so that FMLI does not expand **$m0** through **$m9** at parse time. This feature gives **regex** much of the power of **cut**(1), **paste**(1), and **grep**(1), and some of the capabilities of **sed**(1). If there is no *template,* the default is **$m0$m1$m2$m3$m4$m5$m6$m7$m8$m9 .**

OPTIONS

–e	Evaluate the corresponding template and write the result to *the standard output.*
–v "*string*"	Use *string* instead of *the standard input* to match against patterns.

EXAMPLES

To cut the 4th through 8th letters out of a string (this example will output **strin** and return TRUE):

```
`regex –v "my string is nice" ’^.{3}(.{5})$0’ ’$m0’`
```

In a form, to validate input to field 5 as an integer:

```
valid=`regex –v "$F5" ’^[0-9]+$’`
```

In a form, to translate an environment variable which contains one of the numbers **1**, **2**, **3**, **4**, **5** to the letters **a**, **b**, **c**, **d**, **e**:

```
value=`regex –v "$VAR1" 1 a 2 b 3 c 4 d 5 e ’.*’ ’Error’`
```

Note the use of the pattern **’.*’** to mean "anything else".

In the example below, all three lines constitute a single backquoted expression. This expression, by itself, could be put in a menu definition file. Since backquoted expressions are expanded as they are parsed, and output from a backquoted expression (the **cat** command, in this example) becomes part of the definition file being parsed, this expression would read **/etc/passwd** and make a dynamic menu of all the login ids on the system.

```
`cat /etc/passwd | regex ’^([^:]*)$0.*$’ ’
name=$m0
action=`message "$m0 is a user"`’`
```

DIAGNOSTICS

If none of the patterns match, **regex** returns FALSE, otherwise TRUE.

NOTES

Patterns and templates must often be enclosed in single quotes to turn off the special meanings of characters. Especially if you use the **$m0** through **$m9** variables in the template, since FMLI will expand the variables (usually to "") before **regex** even sees them.

Single characters in character classes (inside **[ ]**) must be listed before character ranges, otherwise they will not be recognized. For example, **[a-zA-Z_/]** will not find underscores (_) or slashes (/), but **[_/a-zA-Z]** will.

The regular expressions accepted by **regcmp** differ slightly from other utilities (that is, **sed**, **grep**, **awk**, **ed**, etc.).

regex with the –**e** option forces subsequent commands to be ignored. In other words if a backquoted statement appears as follows:

```
`regex -e ...; command1; command2`
```

command1 and *command2* would never be executed. However, dividing the expression into two:

```
`regex -e ...``command1; command2`
```

would yield the desired result.

SEE ALSO

awk(1), **cut**(1), **grep**(1), **paste**(1), **sed**(1), **regcmp**(3G)

NAME

reinit – runs an initialization file

SYNOPSIS

reinit *filename*

DESCRIPTION

The **reinit** command is used to change the values of descriptors defined in the initialization file that was named when **fmli** was invoked and / or define additional descriptors. FMLI will parse and evaluate the descriptors in *filename*, and then continue running the current application. The argument *filename* must be the name of a valid FMLI initialization file.

The **reinit** command does not re-display the introductory frame or change the layout of screen labels for function keys.

NAME

renice – alter priority of running processes

SYNOPSIS

/usr/ucb/renice *priority pid* . . .

/usr/ucb/renice *priority* [**–p** *pid* . . .] [**–g** *pgrp* . . .] [**–u** *username* . . .]

AVAILABILITY

SUNWscpu

DESCRIPTION

The **renice** command alters the scheduling priority of one or more running processes. By default, the processes to be affected are specified by their process IDs. *priority* is the new priority value.

Users other than the privileged user may only alter the priority of processes they own, and can only monotonically increase their “nice value” within the range 0 to 20. This prevents overriding administrative fiats. The privileged user may alter the priority of any process and set the priority to any value in the range –20 to 20. Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to), 0 (the “base” scheduling priority) and any negative value (to make things go very fast).

If only the priority is specified, the current process (alternatively, process group or user) is used.

OPTIONS

–p *pid* **. . .**	Specify a list of process IDs.
–g *pgrp* **. . .**	Specify a list of process group IDs. The processes in the specified process groups have their scheduling priority altered.
–u *user* **. . .**	Specify a list of user IDs or usernames. All processes owned by each *user* have their scheduling altered.

FILES

/etc/passwd	map user names to user ID’s

SEE ALSO

priocntl(1)

NOTES

If you make the priority very negative, then the process cannot be interrupted.

To regain control you must make the priority greater than zero.

Users other than the privileged user cannot increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

The **priocntl** command subsumes the function of **renice .**

NAME

repeat – shell built-in function to execute a command more than once

SYNOPSIS

csh

repeat *count command*

DESCRIPTION

csh

Repeat *command count* times. *command* is restricted to a one-line statement.

SEE ALSO

csh(1)

NAME

reset – reset the current form field to its default values

SYNOPSIS

reset

DESCRIPTION

The **reset** function changes the entry in a field of a form to its default value; that is, the value displayed when the form was opened.

NAME

rlogin – remote login

SYNOPSIS

rlogin [**–L**] [**–8**] [**–e***c*] [**–l** *username*] *hostname*

AVAILABILITY

SUNWcsu

DESCRIPTION

rlogin establishes a remote login session from your terminal to the remote machine named *hostname*.

Hostnames are listed in the *hosts* database, which may be contained in the **/etc/hosts** file, the Network Information Service (NIS) **hosts** map, the Internet domain name server, or a combination of these. Each host has one official name (the first name in the database entry), and optionally one or more nicknames. Either official hostnames or nicknames may be specified in *hostname*.

Each remote machine may have a file named **/etc/hosts.equiv** containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may **rlogin** from the machines listed in the remote machine's **/etc/hosts.equiv** file without supplying a password. Individual users may set up a similar private equivalence list with the file **.rhosts** in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a space. An entry in a remote user's **.rhosts** file permits the user named *username* who is logged into *hostname* to log in to the remote machine as the remote user without supplying a password. If the name of the local host is not found in the **/etc/hosts.equiv** file on the remote machine, and the local username and hostname are not found in the remote user's **.rhosts** file, then the remote machine will prompt for a password. Hostnames listed in **/etc/hosts.equiv** and **.rhosts** files must be the official hostnames listed in the hosts database; nicknames may not be used in either of these files.

For security reasons, the **.rhosts** file must be owned by either the remote user or by root.

The remote terminal type is the same as your local terminal type (as given in your environment **TERM** variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the remote login is transparent. Flow control using **Ctrl-S** and **Ctrl-Q** and flushing of input and output on interrupts are handled properly.

OPTIONS

–L	Allow the **rlogin** session to be run in “litout” mode.
–8	Pass eight-bit data across the net instead of seven-bit data.
–e*c*	Specify a different escape character, *c*, for the line used to disconnect from the remote host.
–l *username*	Specify a different *username* for the remote login. If you do not use this option, the remote username used is the same as your local username.

Escape Sequences

Lines that you type which start with the tilde character are “escape sequences” (the escape character can be changed using the –**e** options):

˜. Disconnect from the remote host — this is not the same as a logout, because the local host breaks the connection with no warning to the remote end.

˜susp Suspend the login session (only if you are using a shell with Job Control). **susp** is your “suspend” character, usually Control-Z; see **tty**(1).

˜dsusp Suspend the input half of the login, but output will still be seen (only if you are using a shell with Job Control). **dsusp** is your “deferred suspend” character, usually Control-Y; see **tty**(1).

FILES

/etc/passwd	
/usr/hosts/∗	for *hostname* version of the command
/etc/hosts.equiv	list of trusted hostnames with shared usernames
$HOME/.rhosts	private list of trusted hostname / username combinations

SEE ALSO

rsh(1), **stty**(1), **tty**(1), **in.named**(1M), **hosts**(4), **hosts.equiv**(4)

NOTES

When a system is listed in **hosts.equiv**, its security must be as good as local security. One insecure system listed in **hosts.equiv** can compromise the security of the entire system.

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed.

This implementation can only use the TCP network service.

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [–**f**] [–**i**] *filename*. . .

rm –**r** [–**f**] [–**i**] *dirname*. . . [*filename*. . .]

rmdir [–**p**] [–**s**] *dirname*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

rm removes the entries for one or more files from a directory. If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted, otherwise the file remains.

If *filename* is a symbolic link, the link will be removed, but the file or directory to which it refers will not be deleted. A user does not need write permission on a symbolic link to remove it, provided they have write permissions in the directory.

Note: If the standard input is not a terminal, the command will operate as if the –**f** option is in effect.

OPTIONS

The following options apply to **rm**:

–f Remove all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory is attempted, this option will not suppress an error message.

–i Interactive. With this option, **rm** prompts for confirmation before removing any write-protected files. It overrides the –**f** option and remains in effect even if the standard input is not a terminal.

–r Recursively remove directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note: The user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the –**f** option is used, or if the standard input is not a terminal and the –**i** option is not used.

Symbolic links that are encountered with this option will not be traversed.

If the removal of a non-empty, write-protected directory is attempted, the command will always fail (even if the –**f** option is used), resulting in an error message.

The following options apply to **rmdir**:

–p Allow users to remove the directory *dirname* and its parent directories which become empty. A message is printed on the standard output about whether the whole path is removed or part of the path remains for some reason.

–s Suppress the message printed on the standard error when **–p** is in effect.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **rm** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **rm** behaves.

LC_CTYPE
Determines how **rm** handles characters. When **LC_CTYPE** is set to a valid value, **rm** can display and handle text and filenames containing valid characters for that locale. **rm** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **rm** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

rmdir(2), **unlink**(2), **environ**(5)

DIAGNOSTICS

All messages are generally self-explanatory.

It is forbidden to remove the files "**.**" and "**..**" in order to avoid the consequences of inadvertently doing something like the following:

```
rm –r .*
```

Both **rm** and **rmdir** return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

NOTES

A –– permits the user to mark explicitly the end of any command line options, allowing **rm** to recognize filename arguments that begin with a –. As an aid to BSD migration, **rm** will accept – as a synonym for ––. This migration aid may disappear in a future release. If a –– and a – both appear on the same command line, the second will be interpreted as a filename.

NAME

roffbib – format and print a bibliographic database

SYNOPSIS

roffbib [–**e**] [–**h**] [–**m** *filename*] [–**n***p*] [–**o***list*] [–**Q**] [–**r***aN*] [–**s***N*] [–**T***term*] [–**V**] [–**x**] [*filename*] . . .

AVAILABILITY

SUNWdoc

DESCRIPTION

roffbib prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with **sortbib**(1):

```
example% sortbib database | roffbib
```

OPTIONS

roffbib accepts all options understood by **nroff**(1) except –**i** and –**q**.

–**e** Produce equally-spaced words in adjusted lines using full terminal resolution.

–**h** Use output tabs during horizontal spacing to speed output and reduce output character count. TAB settings are assumed to be every 8 nominal character widths.

–**m** *filename*
Prepend the macro file **/usr/share/lib/tmac/tmac.name** to the input files. There should be a space between the –**m** and the macro filename. This set of macros will replace the ones defined in **/usr/share/lib/tmac/tmac.bib**.

–**n***p* Number first generated page *p*.

–**o***list* Print only page numbers that appear in the comma-separated *list* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; a final *N*– means from page *N* to end.

–**Q** Queue output for the phototypesetter. Page offset is set to 1 inch.

–**r***aN* Set register *a* (one-character) to *N*. The command-line argument –**rN1** will number the references starting at 1.

Four command-line registers control formatting style of the bibliography, much like the number registers of **ms**(5). The flag –**rV2** will double space the bibliography, while –**rV1** will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the –**rL6i** argument, and the page offset can be set from the default of 0 to one inch by specifying –**rO1i** (capital O, not zero).

–**s***N* Halt prior to every *N* pages for paper loading or changing (default *N* =1). To resume, enter NEWLINE or RETURN.

–**T***term* Specify *term* as the terminal type.

–**V** Send output to the Versatec. Page offset is set to 1 inch.

–**x** If abstracts or comments are entered following the **%X** field key, **roffbib** will format them into paragraphs for an annotated bibliography. Several **%X** fields may be given if several annotation paragraphs are desired.

FILES

/usr/share/lib/tmac/tmac.bib	file of macros used by **nroff/troff**

SEE ALSO

addbib(1), **indxbib**(1), **lookbib**(1), **nroff**(1) **refer**(1), **sortbib**(1), **troff**(1)

BUGS

Users have to rewrite macros to create customized formats.

NAME

rpcgen – an RPC protocol compiler

SYNOPSIS

rpcgen *infile*

rpcgen [–**a**] [–**A**] [–**b**] [–**C**] [–**D** *name* [= *value*]] [–**i** *size*] [–**I** [–**K** *seconds*]]
[–**L**] [–**M**] [–**N**] [–**T**] [–**Y** *pathname*] *infile*

rpcgen [–**c** | –**h** | –**l** | –**m** | –**t** | –**Sc** | –**Ss** | –**Sm**] [–**o** *outfile*] [*infile*]

rpcgen [–**s** *nettype*] [–**o** *outfile*] [*infile*]

rpcgen [–**n** *netid*] [–**o** *outfile*] [*infile*]

AVAILABILITY

SUNWcsu

DESCRIPTION

rpcgen is a tool that generates C code to implement an RPC protocol. The input to **rpcgen** is a language similar to C known as RPC Language (Remote Procedure Call Language).

rpcgen is normally used as in the first synopsis where it takes an input file and generates three output files. If the *infile* is named **proto.x**, then **rpcgen** generates a header in **proto.h**, XDR routines in **proto_xdr.c**, server-side stubs in **proto_svc.c**, and client-side stubs in **proto_clnt.c**. With the –**T** option, it also generates the RPC dispatch table in **proto_tbl.i**.

rpcgen can also generate sample client and server files that can be customized to suit a particular application. The –**Sc**, –**Ss** and –**Sm** options generate sample client, server and makefile, respectively. The –**a** option generates all files, including sample files. If the infile is **proto.x**, then the client side sample file is written to **proto_client.c**, the server side sample file to **proto_server.c** and the sample makefile to **makefile.proto**.

The server created can be started both by the port monitors (for example, **inetd** or **listen**) or by itself. When it is started by a port monitor, it creates servers only for the transport for which the file descriptor **0** was passed. The name of the transport must be specified by setting up the environment variable **PM_TRANSPORT**. When the server generated by **rpcgen** is executed, it creates server handles for all the transports specified in **NETPATH** environment variable, or if it is unset, it creates server handles for all the visible transports from **/etc/netconfig** file. Note: the transports are chosen at run time and not at compile time. When the server is self-started, it backgrounds itself by default. A special define symbol **RPC_SVC_FG** can be used to run the server process in foreground.

The second synopsis provides special features which allow for the creation of more sophisticated RPC servers. These features include support for user provided **#defines** and RPC dispatch tables. The entries in the RPC dispatch table contain:

- pointers to the service routine corresponding to that procedure,
- a pointer to the input and output arguments
- the size of these routines

A server can use the dispatch table to check authorization and then to execute the service routine; a client library may use it to deal with the details of storage management and XDR data conversion.

The other three synopses shown above are used when one does not want to generate all the output files, but only a particular one. See the EXAMPLES section below for examples of **rpcgen** usage. When **rpcgen** is executed with the –**s** option, it creates servers for that particular class of transports. When executed with the –**n** option, it creates a server for the transport specified by *netid*. If *infile* is not specified, **rpcgen** accepts the standard input.

The C preprocessor, **cc** –**E** is run on the input file before it is actually interpreted by **rpcgen**. For each type of output file, **rpcgen** defines a special preprocessor symbol for use by the **rpcgen** programmer:

RPC_HDR	defined when compiling into headers
RPC_XDR	defined when compiling into XDR routines
RPC_SVC	defined when compiling into server-side stubs
RPC_CLNT	defined when compiling into client-side stubs
RPC_TBL	defined when compiling into RPC dispatch tables

Any line beginning with ‘‘**%**’’ is passed directly into the output file, uninterpreted by **rpcgen**. To specify the path name of the C preprocessor use –**Y** flag.

For every data type referred to in *infile*, **rpcgen** assumes that there exists a routine with the string **xdr_** prepended to the name of the data type. If this routine does not exist in the RPC / XDR library, it must be provided. Providing an undefined data type allows customization of XDR routines.

OPTIONS

–**a**	Generate all files, including sample files.
–**A**	Enable the Automatic MT mode in the server main program. In this mode, the RPC library automatically creates threads to service client requests. This option generates multithread-safe stubs by implicitly turning on the **-M** option. Server multithreading modes and parameters can be set using the **rpc_control**() call. **rpcgen** generated code does not change the default values for the Automatic MT mode.
–**b**	Backward compatibility mode. Generate transport specific RPC code for older versions of the operating system.
–**c**	Compile into XDR routines.
–**C**	Generate header and stub files which can be used with ANSI C compilers. Headers generated with this flag can also be used with C++ programs.
–**D***name***[=***value***]**	Define a symbol *name*. Equivalent to the **#define** directive in the source. If no *value* is given, *value* is defined as **1**. This option may be specified more than once.
–**h**	Compile into **C** data-definitions (a header). –**T** option can be used in conjunction to produce a header which supports RPC dispatch tables.

–i *size* Size at which to start generating inline code. This option is useful for optimization. The default size is 5.

–I Compile support for **inetd**(1M) in the server side stubs. Such servers can be self-started or can be started by **inetd**. When the server is self-started, it backgrounds itself by default. A special define symbol **RPC_SVC_FG** can be used to run the server process in foreground, or the user may simply compile without the –**I** option.

If there are no pending client requests, the **inetd** servers exit after 120 seconds (default). The default can be changed with the –**K** option. All of the error messages for **inetd** servers are always logged with **syslog**(3).

Note: This option is supported for backward compatibility only. It should always be used in conjunction with the –**b** option which generates backward compatibility code. By default (i.e., when –**b** is not specified), **rpcgen** generates servers that can be invoked through portmonitors.

–K *seconds* By default, services created using **rpcgen** and invoked through port monitors wait **120** seconds after servicing a request before exiting. That interval can be changed using the –**K** flag. To create a server that exits immediately upon servicing a request, use –**K 0**. To create a server that never exits, the appropriate argument is –**K** –**1**.

When monitoring for a server, some portmonitors, like **listen**(1M), *always* spawn a new process in response to a service request. If it is known that a server will be used with such a monitor, the server should exit immediately on completion. For such servers, **rpcgen** should be used with –**K 0**.

–l Compile into client-side stubs.

–L When the servers are started in foreground, use **syslog**(3) to log the server errors instead of printing them on the standard error.

–m Compile into server-side stubs, but do not generate a “main” routine. This option is useful for doing callback-routines and for users who need to write their own “main” routine to do initialization.

–M Generate multithread-safe stubs for passing arguments and results between rpcgen generated code and user written code. This option is useful for users who want to use threads in their code.

–N This option allows procedures to have multiple arguments. It also uses the style of parameter passing that closely resembles C. So, when passing an argument to a remote procedure, you do not have to pass a pointer to the argument, but can pass the argument itself. This behavior is different from the old style of **rpcgen** generated code. To maintain backward compatibility, this option is not the default.

–n *netid*	Compile into server-side stubs for the transport specified by *netid*. There should be an entry for *netid* in the netconfig database. This option may be specified more than once, so as to compile a server that serves multiple transports.
–o *outfile*	Specify the name of the output file. If none is specified, standard output is used (–**c**, –**h**, –**l**, –**m**, –**n**, –**s**, –**Sc**, –**Sm**, –**Ss**, and –**t** modes only).
–s *nettype*	Compile into server-side stubs for all the transports belonging to the class *nettype*. The supported classes are **netpath**, **visible**, **circuit_n**, **circuit_v**, **datagram_n**, **datagram_v**, **tcp**, and **udp** (see **rpc**(3N) for the meanings associated with these classes). This option may be specified more than once. Note: the transports are chosen at run time and not at compile time.
–Sc	Generate sample client code that uses remote procedure calls.
–Sm	Generate a sample Makefile which can be used for compiling the application.
–Ss	Generate sample server code that uses remote procedure calls.
–t	Compile into RPC dispatch table.
–T	Generate the code to support RPC dispatch tables.
	The options –**c**, –**h**, –**l**, –**m**, –**s**, –**Sc**, –**Sm**, –**Ss**, and –**t** are used exclusively to generate a particular type of file, while the options –**D** and –**T** are global and can be used with the other options.
–Y *pathname*	Give the name of the directory where **rpcgen** will start looking for the C-preprocessor.

EXAMPLES

The following example:

```
example% rpcgen –T prot.x
```

generates all the five files: **prot.h**, **prot_clnt.c**, **prot_svc.c**, **prot_xdr.c** and **prot_tbl.i**.

The following example sends the C data-definitions (header) to the standard output.

```
example% rpcgen –h prot.x
```

To send the test version of the **-DTEST**, server side stubs for all the transport belonging to the class **datagram_n** to standard output, use:

```
example% rpcgen –s datagram_n –DTEST prot.x
```

To create the server side stubs for the transport indicated by *netid* **tcp**, use:

```
example% rpcgen –n tcp –o prot_svc.c
```

SEE ALSO

cc(1B), **inetd**(1M), **listen**(1M), **syslog**(3), **rpc**(3N), **rpc_svc_calls**(3N)

The **rpcgen** chapter in the *Network Interfaces Programmer's Guide* manual.

NAME

rsh, remsh, remote_shell – remote shell

SYNOPSIS

rsh [–**n**] [–**l** *username*] *hostname command*

rsh *hostname* [–**n**] [–**l** *username*] *command*

remsh [–**n**] [**-l** *username*] *hostname command*

remsh *hostname* [–**n**] [–**l** *username*] *command*

hostname [–**n**] [–**l** *username*] *command*

AVAILABILITY

SUNWcsu

DESCRIPTION

rsh connects to the specified *hostname* and executes the specified *command*. **rsh** copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; **rsh** normally terminates when the remote command does.

If you omit *command*, instead of executing a single command, **rsh** logs you in on the remote host using **rlogin**(1). Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. See EXAMPLES.

Hostnames are given in the *hosts* database, which may be contained in the **/etc/hosts** file, the Internet domain name database, or both. Each host has one official name (the first name in the database entry) and optionally one or more nicknames. Official hostnames or nicknames may be given as *hostname*.

If the name of the file from which **rsh** is executed is anything other than **rsh**, **rsh** takes this name as its *hostname* argument. This allows you to create a symbolic link to **rsh** in the name of a host which, when executed, will invoke a remote shell on that host. By creating a directory and populating it with symbolic links in the names of commonly used hosts, then including the directory in your shell's search path, you can run **rsh** by typing *hostname* to your shell.

If **rsh** is invoked with the basename **remsh**, **rsh** will check for the existence of the file **/usr/bin/remsh**. If this file exists, **rsh** will behave as if **remsh** is an alias for **rsh**. If **/usr/bin/remsh** does not exist, **rsh** will behave as if **remsh** is a host name.

Each remote machine may have a file named **/etc/hosts.equiv** containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may **rsh** from the machines listed in the remote machine's **/etc/hosts** file. Individual users may set up a similar private equivalence list with the file **.rhosts** in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a space. The entry permits the user named *username* who is logged into *hostname* to use **rsh** to access the remote machine as the remote user. If the name of the local host is not found in the **/etc/hosts.equiv** file on the remote machine, and the local username and hostname are not found in the remote user's **.rhosts** file, then the access is denied. The hostnames listed in the **/etc/hosts.equiv** and **.rhosts** files must be the official hostnames listed in the **hosts** database; nicknames may not be used in either of these files.

rsh will not prompt for a password if access is denied on the remote machine unless the *command* argument is omitted.

OPTIONS

–**l** *username*	Use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username.
–**n**	Redirect the input of **rsh** to **/dev/null**. You sometimes need this option to avoid unfortunate interactions between **rsh** and the shell which invokes it. For example, if you are running **rsh** and invoke a **rsh** in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. The –**n** option will prevent this.

The type of remote shell (**sh**, **rsh**, or other) is determined by the user’s entry in the file **/etc/passwd** on the remote system.

EXIT CODES

Returns **0** upon successful completion, **1** otherwise.

EXAMPLES

The following command:

```
example% rsh lizard cat lizard.file >> example.file
```

appends the remote file **lizard.file** from the machine called ‘‘lizard’’ to the file called **example.file** on the machine called ‘‘example,’’ while the command:

```
example% rsh lizard cat lizard.file ">>" lizard.file2
```

appends the file **lizard.file** on the machine called ‘‘lizard’’ to the file **another.lizard.file** which also resides on the machine called ‘‘lizard.’’

FILES

/etc/hosts
/etc/passwd

SEE ALSO

rlogin(1), **vi**(1), **in.named**(1M), **hosts**(4), **hosts.equiv**(4)

NOTES

When a system is listed in **hosts.equiv**, its security must be as good as local security. One insecure system listed in **hosts.equiv** can compromise the security of the entire system.

You cannot run an interactive command (such as **vi**(1)); use **rlogin** if you wish to do so.

Stop signals stop the local **rsh** process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

The current local environment is not passed to the remote shell.

Sometimes the –**n** option is needed for reasons that are less than obvious. For example, the command:

```
example% rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf –
```

will put your shell into a strange state. Evidently, what happens is that the **tar** terminates before the **rsh**. The **rsh** then tries to write into the ‘‘broken pipe’’ and, instead of terminating neatly, proceeds to compete with your shell for its standard input. Invoking **rsh** with the –**n** option avoids such incidents.

This bug occurs only when **rsh** is at the beginning of a pipeline and is not reading standard input. Do not use the –**n** if **rsh** actually needs to read standard input. For example,

```
example% tar cf – . | rsh sundial dd of=/dev/rmt0 obs=20b
```

does not produce the bug. If you were to use the –**n** in a case like this, **rsh** would incorrectly read from **/dev/null** instead of from the pipe.

NAME

run – run an executable

SYNOPSIS

run [**g–s**] [–**e**] [–**n**] [–**t** *string*] *program*

DESCRIPTION

The **grun** function runs *program*, using the PATH variable to find it. By default, when *program* has completed, the user is prompted (**Press ENTER to continue:**), before being returned to FMLI. The argument *program* is a system executable followed by its options (if any).

OPTIONS

g–**e**	If **g-e** is specified the user will be prompted before returning to FMLI only if there is an error condition
g–**n**	If **g-n** is specified the user will never be prompted before returning to FMLI (useful for programs like **gvi** , in which the user must do some specific action to exit in the first place).
g–**s**	The **g-s** option means "silent", implying that the screen will not have to be repainted when *program* has completed. NOTE: The **g-s** option should only be used when *program* does not write to the terminal. In addition, when **g**–**s** is used, *program* cannot be interrupted, even if it recognizes interrupts.
g–**t***string*	If **g**–**t** is specified, *string* is the name this process will have in the pop-up menu generated by the **gfrm-list** command. This feature requires the executable **gfacesuspend** , (See **face**(1)), to suspend the process and return to the FMLI application.

EXAMPLE

Here is a menu that uses **grun**:

```
gmenu="Edit special System files"

name="Password file"
action=`run –e vi /etc/passwd`

name="Group file"
action=`run –e vi /etc/group`

name="My .profile"
action=`run –n vi $HOME/.profile`
```

NAME

rup – show host status of remote machines (RPC version)

SYNOPSIS

rup [–**hlt**]
rup [*host*. . .]

AVAILABILITY

SUNWesu

DESCRIPTION

rup gives a status similar to **uptime** for remote machines. It broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

When *host* arguments are given, rather than broadcasting **rup** will only query the list of specified hosts.

A remote host will only respond if it is running the **rstatd** daemon, which is normally started up from **inetd**(1M).

OPTIONS

–**h**	Sort the display alphabetically by host name.
–**l**	Sort the display by load average.
–**t**	Sort the display by up time.

FILES

/etc/servers

SEE ALSO

ruptime(1), **inetd**(1M)

SPARC: Installing Solaris Software
x86: Installing Solaris Software

BUGS

Broadcasting does not work through gateways.

NAME

rup – show host status of remote machines (RPC version)

SYNOPSIS

rup [–**hlt**]
rup [*host*. . .]

DESCRIPTION

rup gives a status similar to **uptime** for remote machines. It broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

When *host* arguments are given, rather than broadcasting **rup** only queries the list of specified hosts.

A remote host will only respond if it is running the **rstatd** daemon, which is normally started up from **inetd**(1M).

OPTIONS

–**h**	Sort the display alphabetically by host name.
–**l**	Sort the display by load average.
–**t**	Sort the display by up time.

SEE ALSO

ruptime(1), **inetd**(1M)

BUGS

Broadcasting does not work through gateways.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**–alrtu**]

AVAILABILITY

SUNWcsu

DESCRIPTION

ruptime gives a status line like **uptime** for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name, but this order can be changed by specifying one of the options listed below.

OPTIONS

–a	Count even those users who have been idle for an hour or more.
–l	Sort the display by load average.
–r	Reverse the sorting order.
–t	Sort the display by up time.
–u	Sort the display by number of users.

FILES

/var/spool/rwho/whod.*	data files

SEE ALSO

rwho(1), **in.rwhod**(1M)

NAME

rusage – print resource usage for a command

SYNOPSIS

/usr/ucb/rusage *command*

AVAILABILITY

SUNWscpu

DESCRIPTION

The **rusage** command is similar to **time**(1). It runs the given *command*, which must be specified; that is, *command* is not optional as it is in the C shell's timing facility. When the command is complete, **rusage** displays the real (wall clock), the system CPU, and the user CPU times which elapsed during execution of the command, plus other fields in the **rusage** structure, all on one long line. Times are reported in seconds and hundredths of a second.

EXAMPLES

The example below shows the format of **rusage** output.

```
example% rusage wc /usr/share/man/man1/csh (1)
3045   13423   78071 /usr/share/man/man1/csh (1)
2.26 real 0.80 user 0.36 sys 11 pf 38 pr 0 sw 11 rb 0 wb 16 vcx 37 icx 24 mx 0 ix 1230 id 9 is
example%
```

Each of the fields identified corresponds to an element of the **rusage** structure, as described in **getrusage**(3B), as follows:

real		**elapsed real time**
user	**ru_utime**	**user time used**
sys	**ru_stime**	**system time used**
pf	**ru_majflt**	**page faults requiring physical I/O**
pr	**ru_minflt**	**page faults not requiring physical I/O**
sw	**ru_nswap**	**swaps**
rb	**ru_inblock**	**block input operations**
wb	**ru_oublock**	**block output operations**
vcx	**ru_nvcsw**	**voluntary context switches**
icx	**ru_nivcsw**	**involuntary context switches**
mx	**ru_maxrss**	**maximum resident set size**
ix	**ru_ixrss**	**currently 0**
id	**ru_idrss**	**integral resident set size**
is	**ru_isrss**	**currently 0**

SEE ALSO

csh(1), **time**(1), **getrusage**(3B)

BUGS

When the command being timed is interrupted, the timing values displayed may be inaccurate.

NAME

rusers – who's logged in on remote machines

SYNOPSIS

rusers [**–ahilu**] *host* . . .

AVAILABILITY

SUNWesu

DESCRIPTION

The **rusers** command produces output similar to **who**(1), but for remote machines. The listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

The default is to print out the names of the users logged in. When the **–l** flag is given, additional information is printed for each user:

*userid hostname***:***terminal login date login time idle time login host*

If *hostname* and *login host* are the same value, the *login host* field is not displayed. Likewise, if *hostname* is not idle, the *idle time* is not displayed.

A remote host will only respond if it is running the **rusersd** daemon, which may be started up from **inetd**(1M) or **listen**(1M).

OPTIONS

–a	Give a report for a machine even if no users are logged on.
–h	Sort alphabetically by host name.
–i	Sort by idle time.
–l	Give a longer listing in the style of **who**(1).
–u	Sort by number of users.

SEE ALSO

who(1), **inetd**(1M), **listen**(1M), **pmadm**(1M), **sacadm**(1M)

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [–**a**]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **rwho** command produces output similar to **who**(1), but for all machines on your network. If no report has been received from a machine for 5 minutes, **rwho** assumes the machine is down, and does not report users last known to be logged into that machine.

If a user has not typed to the system for a minute or more, **rwho** reports this idle time. If a user has not typed to the system for an hour or more, the user is omitted from the output of **rwho** unless the –**a** flag is given.

OPTIONS

–**a**	Report all users whether or not they have typed to the system in the past hour.

FILES

/var/spool/rwho/whod.*	information about other machines

SEE ALSO

finger(1), **ruptime**(1), **who**(1), **in.rwhod**(1M)

NOTES

rwho does not work through gateways.

The directory **/var/spool/rwho** must exist on the host from which **rwho** is run.

This service takes up progressively more network bandwith as the number of hosts on the local net increases. For large networks, the cost becomes prohibitive.

The **rwho** service daemon, **in.rwhod**(1M), must be enabled for this command to return useful results.

NAME

sag – system activity graph

SYNOPSIS

sag [**–e** *time*] [**–f** *file*] [**–i** *sec*] [**–s** *time*] [**–T** *term*] [**–x** *spec*] [**–y** *spec*]

DESCRIPTION

sag graphically displays the system activity data stored in a binary data file by a previous **sar**(1) run. Any of the **sar** data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. **sag** invokes **sar** and finds the desired data by string-matching the data column header (run **sar** to see what is available).

OPTIONS

These *options* are passed through to **sar**:

–e *time* Select data up to *time*. Default is 18:00.

–f *file* Use *file* as the data source for **sar**. Default is the current daily data file **/usr/adm/sa/sa***dd*.

–i *sec* Select data at intervals as close as possible to *sec* seconds.

–s *time* Select data later than *time* in the form *hh* [:*mm*]. Default is 08:00.

Other *options*:

–T *term* Produce output suitable for terminal *term*. Default for *term* is **$TERM**.

–x *spec* x axis specification with *spec* in the form:

name [*op name*] . . . [*lo hi*]

name is either a string that will match a column header in the **sar** report, with an optional device name in square brackets, for example, **r+w/s[dsk–1]**, or an integer value. *op* is **+** – ∗ or **/** surrounded by blank spaces. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, **+** and – have precedence over ∗ and **/**. Evaluation is left to right. Thus, A / A + B ∗ 100 is evaluated as (A / (A+B))∗100, and A + B / C + D is (A+B) / (C+D). *lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

Enclose *spec* in double-quotes (" ") if it includes white space.

A single *spec* is permitted for the x axis. If unspecified, *time* is used.

–y *spec* y axis specification with *spec* in the same form as for **–x**. Up to 5 *spec*'s separated by a semi-colon (;) may be given for **–y**. The **–y** default is:

–y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLES

To see today's CPU utilization:

```
example% sag
```

To see activity over 15 minutes of all disk drives:

```
example% set TS=`date  +%H:%M`
example% sar  –o  tempfile  60  15
example% set TE=`date  +%H:%M`
example% sag  –f  tempfile  –s  $TS  –e  $TE  –y  "r+w/s[dsk]"
```

FILES

/usr/adm/sa/sa*dd*	daily data file for day *dd*

SEE ALSO

sar(1)

NAME

sar – system activity reporter

SYNOPSIS

sar [–**aAbcdgkmpqruvwy**] [–**o** *filename*] *t* [*n*]

sar [–**aAbcdgkmpqruvwy**] [–**e** *time*] [–**f** *filename*] [–**i** *sec*] [–**s** *time*]

DESCRIPTION

In the first instance **sar** samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If *t* is specified with more than one option, all headers are printed together and the output may be difficult to read. (If the sampling interval is less than 5, the activity of **sar** itself may effect the sample.) If the –**o** option is specified, it saves the samples in *filename* in binary format. The default value of *n* is 1.

In the second instance no sampling interval is specified. **sar** extracts data from a previously recorded *filename*, either the one specified by the –**f** option or, by default, the standard system activity daily data file **/var/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded using the –**e** and –**s** arguments with *time* specified in the form *hh*[:*mm*[:*ss*]]. The –**i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

OPTIONS

The following options modify the subsets of information reported by **sar**.

–**a**	Report use of file access system routines: iget / s, namei / s, dirblk / s.
–**A**	Report all data. Equivalent to –**abcdgkmpqruvwy**.
–**b**	Report buffer activity: bread / s, bwrit / s – transfers per second of data between system buffers and disk or other block devices; lread / s, lwrit / s – accesses of system buffers; %rcache, %wcache – cache hit ratios, that is, (1–bread / lread) as a percentage; pread / s, pwrit / s – transfers using raw (physical) device mechanism.
–**c**	Report system calls: scall / s – system calls of all types; sread / s, swrit / s, fork / s, exec / s – specific system calls; rchar / s, wchar / s – characters transferred by read and write system calls. No incoming or outgoing **exec**(2) and **fork**(2) calls are reported.
–**d**	Report activity for each block device (for example, disk or tape drive) with the exception of XDC disks and tape drives. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is: %busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time; read / s, write / s, blks / s – number of read / write transfers from or to device, number of bytes transferred in 512-byte units; avseek – number of milliseconds per average seek.

For more general system statistics, use **iostat**(1M), **sar**(1M), or **vmstat**(1M).

See *Peripherals Administration* for naming conventions for disks.

−g Report paging activities:
pgout/s – page-out requests per second;
ppgout/s – pages paged-out per second;
pgfree/s – pages per second placed on the free list by the page stealing daemon;
pgscan/s – pages per second scanned by the page stealing daemon.
%ufs_ipf – the percentage of UFS inodes taken off the freelist by iget which had reusable pages associated with them. These pages are flushed and cannot be reclaimed by processes. Thus this is the percentage of igets with page flushes.

−k Report kernel memory allocation (KMA) activities:
sml_mem, alloc, fail – information about the memory pool reserving and allocating space for small requests: the amount of memory in bytes KMA has for the small pool, the number of bytes allocated to satisfy requests for small amounts of memory, and the number of requests for small amounts of memory that were not satisfied (failed);
lg_mem, alloc, fail – information for the large memory pool (analogous to the information for the small memory pool);
ovsz_alloc, fail – the amount of memory allocated for oversize requests and the number of oversize requests which could not be satisfied (because oversized memory is allocated dynamically, there is not a pool).

−m Report message and semaphore activities:
msg/s, sema/s – primitives per second.

−p Report paging activities:
atch/s – page faults per second that are satisfied by reclaiming a page currently in memory (attaches per second);
pgin/s – page-in requests per second;
ppgin/s – pages paged-in per second;
pflt/s – page faults from protection errors per second (illegal access to page) or "copy-on-writes";
vflt/s – address translation page faults per second (valid page not in memory);
slock/s – faults per second caused by software lock requests requiring physical I/O.

−q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc – run queue of processes in memory and runnable;
swpq-sz, %swpocc – these are no longer reported by sar.

−r Report unused memory pages and disk blocks:
freemem – average pages available to user processes;
freeswap – disk blocks available for page swapping.

–u	Report CPU utilization (the default): %usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.
–v	Report status of process, i-node, file tables: proc-sz, inod-sz, file-sz, lock-sz – entries/size for each table, evaluated once at sampling point; ov – overflows that occur between sampling points for each table.
–w	Report system swapping and switching activity: swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs); pswch/s – process switches.
–y	Report TTY device activity: rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate; rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.
–e *time*	Select data up to *time*. Default is 18:00.
–f *filename*	Use *filename* as the data source for **sar**. Default is the current daily data file **/usr/adm/sa/sa***dd*.
–i *sec*	Select data at intervals as close as possible to *sec* seconds.
–o *filename*	Save samples in file, *filename*, in binary format.
–s *time*	Select data later than *time* in the form *hh*[:*mm*]. Default is 08:00.

EXAMPLES

To see today's CPU activity so far:

```
example% sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
example% sar –o temp 60 10
```

To later review disk and tape activity from that period:

```
example% sar –d –f temp
```

FILES

/var/adm/sa/sa*dd*	daily data file, where *dd* are digits representing the day of the month

SEE ALSO

sag(1), **iostat**(1M), **sar**(1M), **vmstat**(1M), **exec**(2), **fork**(2)

Security, Performance, and Accounting Administration

Peripherals Administration

NAME

sccs-admin, admin – create and administer SCCS history files

SYNOPSIS

/usr/ccs/bin/admin [–**bhnz**] [–**a***username* | *groupid*] . . . [–**d***flag*] . . .
[–**e***username* | *groupid*] . . . [–**f***flag* [*value*]] . . . [–**i** [*filename*]]
[–**l a** | *release*[**,***release*. . .]] [–**m** *mr-list*] [–**r***release*]
[–**t** [*description-file*]] [–**y**[*comment*]] *s.filename* . . .

DESCRIPTION

admin creates or modifies the flags and other parameters of SCCS history files. Filenames of SCCS history files begin with the '**s.**' prefix, and are referred to as **s.**files**,** or ''history'' files.

The named **s.**file is created if it does not exist already. Its parameters are initialized or modified according to the options you specify. Parameters not specified are given default values when the file is initialized, otherwise they remain unchanged.

If a directory name is used in place of the *s.filename* argument, the **admin** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error. The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

OPTIONS

–**b** Force encoding of binary data. Files that contain ASCII NUL or other control characters, or that do not end with a NEWLINE, are recognized as binary data files. The contents of such files are stored in the history file in encoded form. See **uuencode**(1C) for details about the encoding. This option is normally used in conjunction with –**i** to force **admin** to encode initial versions not recognized as containing binary data.

–**h** Check the structure of an existing **s.**file (see **sccsfile**(4)), and compare a newly computed check-sum with one stored in the first line of that file. –**h** inhibits writing on the file; and so nullifies the effect of any other options.

–**n** Create a new SCCS history file.

–**z** Recompute the file check-sum and store it in the first line of the **s.**file. Caution: it is important to verify the contents of the history file (see **sccs-val**(1), and the **print** subcommand in **sccs**(1)), since using –**z** on a truly corrupted file may prevent detection of the error.

–**a***username* | *groupid*
Add a user name, or a numerical group ID, to the list of users who may check deltas in or out. If the list is empty, any user is allowed to do so.

–**d***flag* Delete the indicated *flag* from the SCCS file. The –**d** option may be specified only for existing s.files. See –**f** for the list of recognized flags.

–**e***username* | *groupid*
Erase a user name or group ID from the list of users allowed to make deltas.

–f*flag* [*value*]

Set the indicated *flag* to the (optional) *value* specified. The following flags are recognized:

b Enable branch deltas. When **b** is set, branches can be created using the **–b** option of the SCCS **get** command (see **sccs-get**(1)).

c*ceil* Set a ceiling on the releases that can be checked out. *ceil* is a number less than or equal to 9999. If **c** is not set, the ceiling is 9999.

f*floor* Set a floor on the releases that can be checked out. The floor is a number greater than 0 but less than 9999. If **f** is not set, the floor is 1.

d*sid* The default delta number, or SID, to be used by an SCCS **get** command.

i Treat the '**No id keywords (ge6)**' message issued by an SCCS **get** or **delta** command as an error rather than a warning.

j Allow concurrent updates.

la
l*release*[**,***release* . . .]

Lock the indicated list of releases against deltas. If **a** is used, lock out deltas to all releases. An SCCS '**get –e**' command fails when applied against a locked release.

n Create empty releases when releases are skipped. These null (empty) deltas serve as anchor points for branch deltas.

q*value* Supply a *value* to which the **%Q%** keyword is to expand when a read-only version is retrieved with the SCCS **get** command.

m*module*

Supply a value for the module name to which the **%M%** keyword is to expand. If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

t*type* Supply a value for the module type to which the **%Y%** keyword is to expand.

v [*program*]

Specify a validation *program* for the MR numbers associated with a new delta. The optional *program* specifies the name of an MR number validity checking *program*. If this flag is set when creating an SCCS file, the **–m** option must also be used, in which case the list of MRs may be empty.

–**i** [*filename*]
Initialize the history file with text from the indicated file. This text constitutes the initial delta, or set of checked-in changes. If *filename* is omitted, the initial text is obtained from the standard input. Omitting the –**i** option altogether creates an empty **s.**file. You can only initialize one **s.**file with text using –**i**. This option implies the –**n** option.

–**la**

–**l** *release*[**,***release*. . .]
Unlock the specified releases so that deltas can be checked in. If **a** is specified, allow deltas to be checked in for all releases.

–**m** [*mr-list*]
Insert the indicated Modification Request (MR) numbers into the commentary for the initial version. When specifying more than one MR number on the command line, *mr-list* takes the form of a quoted, space-separated list. A warning results if the **v** flag is not set or the MR validation fails.

–**r***release*
Specify the release for the initial delta. –**r** may be used only in conjunction with –**i**. The initial delta is inserted into release 1 if this option is omitted. The level of the initial delta is always 1; initial deltas are named 1.1 by default.

–**t** [*description-file*]
Insert descriptive text from the file *description-file*. When –**t** is used in conjunction with –**n**, or –**i** to initialize a new s.file, the *description-file* must be supplied. When modifying the description for an existing file: a –**t** option without a *description-file* removes the descriptive text, if any; a –**t** option with a *description-file* replaces the existing text.

–**y** [*comment*]
Insert the indicated *comment* in the ‘‘Comments:’’ field for the initial delta. Valid only in conjunction with –**i** or –**n**. If –**y** option is omitted, a default comment line is inserted that notes the date and time the history file was created.

FILES

s.∗	history file
SCCS/s.∗	history file in SCCS subdirectory
z.∗	temporary lock file

SEE ALSO

sccs(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-rmdel**(1), **sccs-val**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

WARNINGS

The last component of all SCCS filenames must have the '**s.**' prefix. New SCCS files are given mode 444 (see **chmod**(1)). All writing done by **admin** is to a temporary file with an **x.** prefix, created with mode 444 for a new SCCS file, or with the same mode as an existing SCCS file. After successful execution of **admin**, the existing **s.** file is removed and replaced with the **x.**file. This ensures that changes are made to the SCCS file only when no errors have occurred.

It is recommended that directories containing SCCS files have permission mode 755, and that the **s.**files themselves have mode 444. The mode for directories allows only the owner to modify the SCCS files contained in the directories, while the mode of the **s.**files prevents all modifications except those performed using SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner to allow use of a text editor. However, extreme care must be taken when doing this. The edited file should *always* be processed by an '**admin** –**h**' to check for corruption, followed by an '**admin** –**z**' to generate a proper check-sum. Another '**admin** –**h**' is recommended to ensure that the resulting **s.**file is valid.

admin also uses a temporary lock **s.**file, starting with the '**z.**' prefix, to prevent simultaneous updates to the **s.**file. See **sccs-get**(1) for further information about the '**z.**file'.

NAME

sccs-cdc, cdc – change the delta commentary of an SCCS delta

SYNOPSIS

/usr/ccs/bin/cdc –**r***sid* [–**m***mr-list*] [–**y** [*comment*]] *s.filename* . . .

DESCRIPTION

cdc annotates the delta commentary for the SCCS delta ID (SID) specified by the –**r** option in each named **s.**file.

If the **v** flag is set in the **s.**file, you can also use **cdc** to update the Modification Request (MR) list.

If you checked in the delta, or, if you own the file and directory and have write permission, you can use **cdc** to annotate the commentary.

Rather than replacing the existing commentary, **cdc** inserts the new comment you supply, followed by a line of the form:

> *** **CHANGED** *** *yy*/*mm*/*dd hh*/*mm*/*ss username*

above the existing commentary.

If a directory is named as the *s.filename* argument, the **cdc** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). If '–' is given as the *s.filename* argument, each line of the standard input is taken as the name of an SCCS history file to be processed, and the –**m** and –**y** options must be used.

OPTIONS

–**r***sid*	Specify the SID of the delta to change.
–**m***mr-list*	Specify one or more MR numbers to add or delete. When specifying more than one MR on the command line, *mr-list* takes the form of a quoted, space-separated list. To delete an MR number, precede it with a **!** character (an empty MR list has no effect). A list of deleted MRs is placed in the comment section of the delta commentary. If –**m** is not used and the standard input is a terminal, **cdc** prompts with **MRs?** for the list (before issuing the **comments?** prompt). –**m** is only useful when the **v** flag is set in the **s.**file. If that flag has a value, it is taken to be the name of a program to validate the MR numbers. If that validation program returns a non-zero exit status, **cdc** terminates and the delta commentary remains unchanged.
–**y**[*comment*]	Use *comment* as the annotation in the delta commentary. The previous comments are retained; the *comment* is added along with a notation that the commentary was changed. A null *comment* leaves the commentary unaffected. If –**y** is not specified and the standard input is a terminal, **cdc** prompts with **comments?** for the text of the notation to be added. An unescaped NEWLINE character terminates the annotation text.

EXAMPLES

The following command:

```
example% cdc –r1.6 –y"corrected commentary" s.program.c
```

produces the following annotated commentary for delta 1.6 in **s.program.c**:

```
D 1.6 88/07/05 23:21:07 username 9 0 00001/00000/00000
MRs:
COMMENTS:
corrected commentary
*** CHANGED *** 88/07/07 14:09:41 username
performance enhancements in main()
```

FILES

z.*file*	temporary lock file

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

NAME

sccs-comb, comb – combine SCCS deltas

SYNOPSIS

/usr/ccs/bin/comb [–**os**] [–**c***sid-list*] [–**p***sid*] *s.filename* . . .

DESCRIPTION

comb generates a shell script (see **sh**(1)) that you can use to reconstruct the indicated **s.**files. This script is written to the standard output.

If a directory name is used in place of the *s.filename* argument, the **comb** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

If no options are specified, **comb** preserves only the most recent (leaf) delta in a branch, and the minimal number of ancestors needed to preserve the history.

OPTIONS

–**o** For each '**get** –**e**' generated, access the reconstructed file at the release of the delta to be created. Otherwise, the reconstructed file is accessed at the most recent ancestor. The use of –**o** may decrease the size of the reconstructed **s.**file. It may also alter the shape of the delta tree of the original file.

–**s** Generate scripts to gather statistics, rather than combining deltas. When run, the shell scripts report: the file name, size (in blocks) after combining, original size (also in blocks), and the percentage size change, computed by the formula:

$$100 * (\mathit{original} - \mathit{combined}) / \mathit{original}$$

This option can be used to calculate the space that will be saved, before actually doing the combining.

–**c***sid-list*
Include the indicated list of deltas. All other deltas are omitted. *sid-list* is a comma-separated list of SCCS delta IDs (SIDs). To specify a range of deltas, use a '–' separator instead of a comma, between two SIDs in the list.

–**p***SID* The SID of the oldest delta to be preserved.

FILES

s.COMB	reconstructed SCCS file
comb?????	temporary file

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

BUGS

comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

sccs-delta, delta – make a delta to an SCCS file

SYNOPSIS

/usr/ccs/bin/delta [**–nps**] [**–g***sid-list*] [**–m***mr-list*] [**–r***sid*] [**–y**[*comment*]] *s.filename* . . .

DESCRIPTION

delta checks in a record of the line-by-line differences made to a checked-out version of a file under SCCS control. These changes are taken from the writable working copy that was retrieved using the SCCS **get** command (see **sccs-get**(1)). This working copy does not have the '**s.**' prefix, and is also referred to as a **g**-file.

If a directory name is used in place of the *s.filename*, argument, the **delta** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.file** per line (requires –**y**, and in some cases, –**m**).

delta may issue prompts on the standard output depending upon the options specified and the flags that are set in the **s.**file (see **sccs-admin**(1), and the –**m** and –**y** options below, for details).

OPTIONS

–n Retain the edited **g**-file, which is normally removed at the completion of processing.

–p Display line-by-line differences (in **diff**(1) format) on the standard output.

–s Silent. Do not display warning or confirmation messages. Do not suppress error messages (which are written to standard error).

–g*sid-list*
Specify a list of deltas to omit when the file is accessed at the SCCS version ID (SID) created by this delta. *sid-list* is a comma-separated list of SIDs. To specify a range of deltas, use a '–' separator instead of a comma, between two SIDs in the list.

–m [*mr-list*]
If the SCCS file has the **v** flag set (see **sccs-admin**(1)), you must supply one or more Modification Request (MR) numbers for the new delta. When specifying more than one MR number on the command line, *mr-list* takes the form of a quoted, space-separated list. If –**m** is not used and the standard input is a terminal, **delta** prompts with **MRs?** for the list (before issuing the **comments?** prompt). If the **v** flag in the **s.**file has a value, it is taken to be the name of a program to validate the MR numbers. If that validation program returns a non-zero exit status, **delta** terminates without checking in the changes.

–r*sid* When two or more versions are checked out, specify the version to check in. This SID value can be either the SID specified on the **get** command line, or the SID of the new version to be checked in as reported by **get**. A diagnostic results if the specified SID is ambiguous, or if one is required but not supplied.

–**y**[*comment*]
Supply a comment for the delta table (version log). A null comment is accepted, and produces an empty commentary in the log. If –**y** is not specified and the standard input is a terminal, **delta** prompts with '**comments?**'. An unescaped NEWLINE terminates the comment.

FILES

d.*file*	temporary file of differences
p.*file*	lock file for a checked-out version
q.*file*	temporary file
s.*file*	SCCS history file
x.*file*	temporary copy of the **s.**file
z.*file*	temporary file

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

WARNINGS

Lines beginning with an ASCII SOH character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see **sccsfile**(4)) and produces an error.

NAME

sccs-get, get – retrieve a version of an SCCS file

SYNOPSIS

/usr/ccs/bin/get [−**begkmnpst**] [−**l** [**p**]] [−**a***sequence*] [−**c***date-time*] [−**G***g-file*]
[−**i***sid-list*] [−**r***sid*] [−**x***sid-list*] *s.filename* . . .

DESCRIPTION

get retrieves a working copy from the SCCS history file, according to the specified options.

For each *s.filename* argument, **get** displays the SCCS delta ID (SID) and number of lines retrieved.

If a directory name is used in place of the *s.filename* argument, the **get** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '−' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

The retrieved file normally has the same filename base as the **s.**file, less the prefix, and is referred to as the **g-**file.

For each file processed, **get** responds (on the standard output) with the SID being accessed, and with the number of lines retrieved from the **s.**file.

OPTIONS

−b Create a new branch. Used with the −**e** option to indicate that the new delta should have an SID in a new branch. Instead of incrementing the level for version to be checked in, **get** indicates in the **p.**file that the delta to be checked in should either initialize a new branch and sequence (if there is no existing branch at the current level), or increment the branch component of the SID. If the **b** flag is not set in the **s.**file, this option is ignored.

−e Retrieve a version for editing. With this option, **get** places a lock on the **s.**file, so that no one else can check in changes to the version you have checked out. If the **j** flag is set in the **s.**file, the lock is advisory: **get** issues a warning message. Concurrent use of '**get** −**e**' for different SIDs is allowed, however, **get** will not check out a version of the file if a writable version is present in the directory. All SCCS file protections stored in the **s.**file, including the release ceiling, floor, and authorized user list, are honored by '**get** −**e**'.

−g Get the SCCS version ID, without retrieving the version itself. Used to verify the existence of a particular SID.

−k Suppress expansion of ID keywords. −**k** is implied by the −**e**.

−m Precede each retrieved line with the SID of the delta in which it was added to the file. The SID is separated from the line with a TAB.

−n Precede each line with the **%M%** ID keyword and a TAB. When both the −**m** and −**n** options are used, the ID keyword precedes the SID, and the line of text.

−p Write the text of the retrieved version to the standard output. All messages that normally go to the standard output are written to the standard error instead.

–s Suppress all output normally written on the standard output. However, fatal error messages (which always go to the standard error) remain unaffected.

–t Retrieve the most recently created (top) delta in a given release (for example: **–r1**).

–l [**p**] Retrieve a summary of the delta table (version log) and write it to a listing file, with the '**l.**' prefix (called '**l.**file'). When **–lp** is used, write the summary onto the standard output.

–a *sequence*

Retrieve the version corresponding to the indicated delta sequence number. This option is used primarily by the SCCS **comb** command (see **sccs-comb**(1)); for users, **–r** is an easier way to specify a version. **–a** supersedes **–r** when both are used.

–c*date-time*

Retrieve the latest version checked in prior to the date and time indicated by the *date-time* argument. *date-time* takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]]. Units omitted from the indicated date and time default to their maximum possible values; that is **–c7502** is equivalent to **–c750228235959**. Any number of non-numeric characters may separate the various 2 digit components. If white-space characters occur, the *date-time* specification must be quoted.

–G*newname*

Use *newname* as the name of the retrieved version.

–i*sid-list*

Specify a list of deltas to include in the retrieved version. The included deltas are noted in the standard output message. *sid-list* is a comma-separated list of SIDs. To specify a range of deltas, use a '–' separator instead of a comma, between two SIDs in the list.

–r*sid* Retrieve the version corresponding to the indicated SID (delta).

The SID for a given delta is a number, in Dewey decimal format, composed of two or four fields: the *release* and *level* fields, and for branch deltas, the *branch* and *sequence* fields. For instance, if **1.2** is the SID, **1** is the release, and **2** is the level number. If **1.2.3.4** is the SID, **3** is the branch and **4** is the sequence number.

You need not specify the entire SID to retrieve a version with **get**. When you omit **–r** altogether, or when you omit both release and level, **get** normally retrieves the highest release and level. If the **d** flag is set to an SID in the **s.**file and you omit the SID, **get** retrieves the default version indicated by that flag.

When you specify a release but omit the level, **get** retrieves the highest level in that release. If that release does not exist, **get** retrieves highest level from the next-highest existing release.

Similarly with branches, if you specify a release, level and branch, **get** retrieves the highest sequence in that branch.

–**x***sid-list*
Exclude the indicated deltas from the retrieved version. The excluded deltas are noted in the standard output message. *sid-list* is a comma-separated list of SIDs. To specify a range of deltas, use a '**-**' separator instead of a comma, between two SIDs in the list.

USAGE

ID Keywords

In the absence of –**e** or –**k**, **get** expands the following ID keywords by replacing them with the indicated values in the text of the retrieved source.

Keyword	*Value*
%A%	Shorthand notation for an ID line with data for **what**(1): **%Z%%Y% %M% %I%%Z%**
%B%	SID branch component
%C%	Current line number. Intended for identifying messages output by the program such as ''*this shouldn't have happened*'' type errors. It is *not* intended to be used on every line to provide sequence numbers.
%D%	Current date: *yy***/***mm***/***dd*
%E%	Date newest applied delta was created: *yy***/***mm***/***dd*
%F%	SCCS **s.**file name
%G%	Date newest applied delta was created: *mm***/***dd***/***yy*
%H%	Current date: *mm***/***dd***/***yy*
%I%	SID of the retrieved version: **%R%.%L%.%B%.%S%**
%L%	SID level component
%M%	Module name: either the value of the **m** flag in the **s.**file (see **sccs-admin**(1)), or the name of the **s.**file less the prefix
%P%	Fully qualified **s.**file name
%Q%	Value of the **q** flag in the **s.**file
%R%	SID Release component
%S%	SID Sequence component
%T%	Current time: *hh***:***mm***:***ss*
%U%	Time the newest applied delta was created: *hh***:***mm***:***ss*
%W%	Shorthand notation for an ID line with data for **what**: **%Z%%M%** **%I%**
%Y%	Module type: value of the **t** flag in the **s.**file
%Z%	4-character string: '**@(#)**', recognized by **what**.

ID String

The table below explains how the SCCS identification string is determined for retrieving and creating deltas.

Determination of SCCS Identification String				
SID* Specified	–**b** Option Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* ‘R’, ‘L’, ‘B’, and ‘S’ are the ‘release’, ‘level’, ‘branch’, and ‘sequence’ components of the SID, respectively; ‘m’ means ‘maximum’. Thus, for example, ‘R.mL’ means ‘the maximum level number within release R’; ‘R.L.(mB+1).1’ means ‘the first sequence number on the *new* branch (that is, maximum branch number plus one) of level L within release R’. Note: if the SID specified is of the form ‘R.L’, ‘R.L.B’, or ‘R.L.B.S’, each of the specified components *must* exist.

** ‘hR’ is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** Forces creation of the *first* delta in a *new* release.

# Successor.

† The –**b** option is effective only if the **b** flag is present in the file. An entry of ‘–’ means ‘irrelevant’.

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

FILES

‘‘g-file’’	version retrieved by **get**
l.*file*	file containing extracted delta table info
p.*file*	permissions (lock) file
z.*file*	temporary copy of **s.***file*

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-sact**(1), **sccs-unget**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, only one file may be named when using –**e**.

NAME

sccs-help, help – ask for help regarding SCCS error or warning messages

SYNOPSIS

/usr/ccs/bin/help [*argument*] . . .

DESCRIPTION

help retrieves information to further explain errors messages and warnings from SCCS commands. It also provides some information about SCCS command usage. If no arguments are given, **help** prompts for one.

An *argument* may be a message number (which normally appears in parentheses following each SCCS error or warning message), or an SCCS command name. **help** responds with an explanation of the message or a usage line for the command.

When all else fails, try '**/usr/ccs/bin/help stuck**'.

FILES

/usr/ccs/lib/help	directory containing files of message text

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **what**(1), **sccsfile**(4)

NAME

sccs-prs, prs – display selected portions of an SCCS history

SYNOPSIS

/usr/ccs/bin/prs [–**ael**] [–**c***date-time*] [–**d***dataspec*] [–**r***sid*] *s.filename* . . .

DESCRIPTION

prs displays part or all of the SCCS file (see **sccsfile**(4)) in a user supplied format.

If a directory name is used in place of the *s.filename* argument, the **prs** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

OPTIONS

In the absence of options, **prs** displays the delta table (version log). In the absence of –**d**, or –**l**, **prs** displays the entry for each delta indicated by the other options.

–**a** Include all deltas, including those marked as removed (see **sccs-rmdel**(1)).

–**e** Request information for all deltas created *earlier* than, and including, the delta indicated with –**r** or –**c**.

–**l** Request information for all deltas created *later* than, and including, the delta indicated with –**r** or –**c**.

–**c***date-time*

Display information on the latest delta checked in prior to the date and time indicated by the *date-time* argument. *date-time* takes the form:

yy[*mm*[*dd*[*hh*[*mm*[*ss*]]]]].

Units omitted from the indicated date and time default to their maximum possible values; that is –**c7502** is equivalent to –**c750228235959**. Any number of non-numeric characters may separate the various 2 digit components. If white-space characters occur, the *date-time* specification must be quoted.

–**d***dataspec*

Produce a report according to the indicated data specification. *dataspec* consists of a (quoted) text string that includes embedded data keywords of the form: '**:***key***:**' (see *Data Keywords*, below). **prs** expands these keywords in the output it produces. To specify a TAB character in the output, use **\t**; to specify a NEWLINE in the output, use **\n**.

–**r***sid* Specify the SCCS delta ID (SID) of the delta for which information is desired. If no SID is specified, the most recently created delta is used.

USAGE

Data Keywords

Data keywords specify which parts of an SCCS file are to be retrieved. All parts of an SCCS file (see **sccsfile**(4)) have an associated data keyword. A data keyword may appear any number of times in a data specification argument to **–d**. These data keywords are listed in the table below:

Keyword	*Data Item*	*File Section**	*Value*	*Format***
:A:	a format for the **what** string:	N/A	**:Z::Y: :M: :I::Z:**	S
:B:	branch number	D	*nnnn*	S
:BD:	body	B	*text*	M
:BF:	branch flag	F	**yes** or **no**	S
:CB:	ceiling boundary	F	**:R:**	S
:C:	comments for delta	D	*text*	M
:D:	date delta created	D	**:Dy:/:Dm:/:Dd:**	S
:Dd:	day delta created	D	*nn*	S
:Dg:	deltas ignored (seq #)	D	**:DS: :DS:**. . .	S
:DI:	seq-no. of deltas included, excluded, ignored	D	**:Dn:/:Dx:/:Dg:**	S
:DL:	delta line statistics	D	**:Li:/:Ld:/:Lu:**	S
:Dm:	month delta created	D	*nn*	S
:Dn:	deltas included (seq #)	D	**:DS: :DS:**. . .	S
:DP:	predecessor delta seq-no.	D	*nnnn*	S
:Ds:	default SID	F	**:I:**	S
:DS:	delta sequence number	D	*nnnn*	S
:Dt:	delta information	D	**:DT: :I: :D: :T: :P: :DS: :DP:**	S
:DT:	delta type	D	**D** or **R**	S
:Dx:	deltas excluded (seq #)	D	**:DS:** . . .	S
:Dy:	year delta created	D	*nn*	S
:F:	**s.**file name	N/A	*text*	S
:FB:	floor boundary	F	**:R:**	S
:FD:	file descriptive text	C	*text*	M
:FL:	flag list	F	*text*	M
:GB:	gotten body	B	*text*	M
:I:	SCCS delta ID (SID)	D	**:R:.:L:.:B:.:S:**	S
:J:	joint edit flag	F	**yes** or **no**	S
:KF:	keyword error/warning flag	F	**yes** or **no**	S
:L:	level number	D	*nnnn*	S
:Ld:	lines deleted by delta	D	*nnnnn*	S
:Li:	lines inserted by delta	D	*nnnnn*	S
:LK:	locked releases	F	**:R:**. . .	S
:Lu:	lines unchanged by delta	D	*nnnnn*	S
:M:	module name	F	*text*	S
:MF:	MR validation flag	F	**yes** or **no**	S
:MP:	MR validation program	F	*text*	S
:MR:	MR numbers for delta	D	*text*	M
:ND:	null delta flag	F	**yes** or **no**	S
:Q:	user defined keyword	F	*text*	S

:P:	user who created delta	D	*username*	S
:PN:	**s.**file's pathname	N/A	*text*	S
:R:	release number	D	*nnnn*	S
:S:	sequence number	D	*nnnn*	S
:T:	time delta created	D	**:Th:::Tm:::Ts:**	S
:Th:	hour delta created	D	*nn*	S
:Tm:	minutes delta created	D	*nn*	S
:Ts:	seconds delta created	D	*nn*	S
:UN:	user names	U	*text*	M
:W:	a form of **what** string	N/A	**:Z::M:\t:I:**	S
:Y:	module type flag	F	*text*	S
:Z:	**what** string delimiter	N/A	**@(#)**	S

*B = body, D = delta table, F = flags, U = user names

**S = simple format, M = multi-line format

EXAMPLES

The following command:

```
example% /usr/ccs/bin/prs –e –d":I:\t:P:" program.c
```

produces:

```
1.6    username
1.5    username
. . .
```

FILES

/tmp/pr?????	temporary file

SEE ALSO

sccs(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prt**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

NAME

sccs-prt, prt – display delta table information from an SCCS file

SYNOPSIS

/usr/ccs/bin/prt [–**abdefistu**] [–**c***date-time*] [–**r***date-time*] [–**y***sid*] *s.filename* . . .

DESCRIPTION

prt prints selected portions of an SCCS file. By default, it prints the delta table (version log).

If a directory name is used in place of the *s.filename* argument, the **prt** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

OPTIONS

If any option other than –**y**, –**c**, or –**r** is supplied, the name of each file being processed (preceded by one NEWLINE and followed by two NEWLINE characters) appears above its contents.

If none of the –**u**, –**f**, –**t**, or –**b** options are used, –**d** is assumed. –**s**, –**i** are mutually exclusive, as are –**c** and –**r**.

–**a** Display log entries for all deltas, including those marked as removed.

–**b** Print the body of the **s.**file.

–**d** Print delta table entries. This is the default.

–**e** Everything. This option implies –**d**, –**i**, –**u**, –**f**, and –**t**.

–**f** Print the flags of each named **s.**file.

–**i** Print the serial numbers of included, excluded, and ignored deltas.

–**s** Print only the first line of the delta table entries; that is, only up to the statistics.

–**t** Print the descriptive text contained in the **s.**file.

–**u** Print the user-names and / or numerical group IDs of users allowed to make deltas.

–**c***date-time*
Exclude delta table entries that are specified cutoff date and time. Each entry is printed as a single line, preceded by the name of the SCCS file. This format (also produced by –**r ,** and –**y**) makes it easy to sort multiple delta tables in chronological order. When both –**y** and –**c**, or –**y** and –**r** are supplied, **prt** stops printing when the first of the two conditions is met.

–**r***date-time*
Exclude delta table entries that are newer than the specified cutoff date and time.

–**y***sid* Exclude delta table entries made prior to the SID specified. If no delta in the table has the specified SID, the entire table is printed. If no SID is specified, the most recent delta is printed.

USAGE

Output Format

The following format is used to print those portions of the **s.**file that are specified by the various options.

- NEWLINE
- Type of delta (**D** or **R**)
- SPACE
- SCCS delta ID (SID)
- TAB
- Date and time of creation in the form: *yy*/*mm*/*dd hh*/*mm*/*ss*
- SPACE
- Username the delta's creator
- TAB
- Serial number of the delta
- SPACE
- Predecessor delta's serial number
- TAB
- Line-by-line change statistics in the form: *inserted*/*deleted*/*unchanged*
- NEWLINE
- List of included deltas, followed by a NEWLINE (only if there were any such deltas and the –**i** options was used)
- List of excluded deltas, followed by a NEWLINE (only if there were any such deltas and the –**i** options was used)
- List of ignored deltas, followed by a NEWLINE (only if there were any such deltas and the –**i** options was used)
- List of modification requests (MR s), followed by a NEWLINE (only if any MR numbers were supplied).
- Lines of the delta commentary (if any), followed by a NEWLINE.

EXAMPLES

The following command:

```
example% /usr/ccs/bin/prt –y program.c
```

produces a one-line display of the delta table entry for the most recent version:

```
s.program.c:  D 1.6   88/07/06 21:39:39 username   5 4 00159/00080/00636. . .
```

SEE ALSO

sccs(1), **sccs-cdc**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(4)

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

NAME

sccs-rmdel, rmdel – remove a delta from an SCCS file

SYNOPSIS

/usr/ccs/bin/rmdel –**r***sid s.filename* . . .

DESCRIPTION

rmdel removes the delta specified by the SCCS delta ID (SID) supplied with –**r**. The delta to be removed must be the most recent (leaf) delta in its branch. In addition, the SID must *not* be that of a version checked out for editing: it must not appear in any entry of the version lock file (**p.**file).

If you created the delta, or, if you own the file and directory and have write permission, you can remove it with **rmdel**.

If a directory name is used in place of the *s.filename* argument, the **rmdel** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

OPTIONS

–**r***sid* Remove the version corresponding to the indicated SID (delta).

FILES

p.*file*	permissions file
s.*file*	history file
z.*file*	temporary copy of the **s.**file

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

NAME

sccs-sact, sact – show editing activity status of an SCCS file

SYNOPSIS

/usr/ccs/bin/sact *s.filename* . . .

DESCRIPTION

sact informs the user of any SCCS files that are checked out for editing.

The output for each named file consists of five fields separated by SPACE characters.

- SID of a delta that currently exists in the SCCS file, to which changes will be made to make the new delta
- SID for the new delta to be created
- Username of the person who has the file checked out for editing.
- Date that the version was checked out.
- Time that the version was checked out.

If a directory name is used in place of the *s.filename* argument, the **sact** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

SEE ALSO

sccs(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

BUGS

sact is not recognized as a subcommand of **sccs**(1).

NAME

sccs-sccsdiff, sccsdiff – compare two versions of an SCCS file

SYNOPSIS

/usr/ccs/bin/sccsdiff [**–p**] **–r***sid* **–r***sid* [*diff-options*] *s.filename*

DESCRIPTION

sccsdiff compares two versions of an SCCS file and displays the differences between the two versions. Any number of SCCS files may be specified; the options specified apply to all named **s.**files.

OPTIONS

–p Pipe output for each file through **pr**(1).

–r*sid* Specify a version corresponding to the indicated SCCS delta ID (SID) for comparison. Versions are passed to **diff**(1) in the order given.

diff-options
Pass options to **diff**(1), including: **–c**, **–e**, **–f**, **–h**, **–b** and **–D**.

FILES

/tmp/get????? temporary files

SEE ALSO

diff(1), **sccs**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

filename: **No differences**
If the two versions are the same.

Use the SCCS **help** command for explanations of other messages (see **sccs-help**(1)).

NAME

sccs-unget, unget – undo a previous get of an SCCS file

SYNOPSIS

/usr/ccs/bin/unget [**–ns**] [**–r***sid*] *s.filename* . . .

DESCRIPTION

unget undoes the effect of a '**get –e**' done prior to the creation of the pending delta.

If a directory name is used in place of the *s.filename* argument, the **unget** command applies to all **s.**files in that directory. Unreadable **s.**files produce an error; processing continues with the next file (if any). The use of '–' as the *s.filename* argument indicates that the names of files are to be read from the standard input, one **s.**file per line.

OPTIONS

–n Retain the retrieved version, which is otherwise removed.

–s Suppress display of the SCCS delta ID (SID).

–r*sid* When multiple versions are checked out, specify which pending delta to abort. A diagnostic results if the specified SID is ambiguous, or if it is necessary but omitted from the command line.

SEE ALSO

sccs(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **what**(1), **sccsfile**(4)

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

NAME

sccs-val, val – validate an SCCS file

SYNOPSIS

/usr/ccs/bin/val –

/usr/ccs/bin/val [–**s**] [–**m** *name*] [–**r***sid*] [–**y** *type*] *s.filename* . . .

DESCRIPTION

val determines if the specified **s.**files files meet the characteristics specified by the indicated arguments. **val** can process up to 50 files on a single command line.

val has a special argument, '–', which reads the standard input until the end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8–bit code upon exit as described below.

The 8-bit code returned by **val** is a disjunction of the possible errors, that is, it can be interpreted as a bit string where (moving from left to right) the bits set are interpreted as follows:

bit 0 = missing file argument
bit 1 = unknown or duplicate option
bit 2 = corrupted **s.**file
bit 3 = can not open file or file not in **s.**file format
bit 4 = the SCCS delta ID (SID) is invalid or ambiguous
bit 5 = the SID does not exist
bit 6 = mismatch between **%Y%** and –**y** argument
bit 7 = mismatch between **%M%** –**m** argument

val can process two or more files on a given command line, and in turn can process multiple command lines (when reading the standard input). In these cases, an aggregate code is returned which is the logical **OR** of the codes generated for each command line and file processed.

OPTIONS

–**s**	Silent. Suppress the normal error or warning messages.
–**m** *name*	Compare *name* with the **%M%** ID keyword in the **s.**file.
–**r***sid*	Check to see if the indicated SID is ambiguous, invalid, or absent from the **s.**file.
–**y** *type*	Compare *type* with the **%Y%** ID keyword.

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

NAME

sccs – front end for the Source Code Control System (SCCS)

SYNOPSIS

sccs [**–r**] [**–d***rootprefix*] [**–p***subdir*] *subcommand* [*option* . . .] [*filename* . . .]

DESCRIPTION

The **sccs** command is a comprehensive, straightforward front end to the various utility programs of the Source Code Control System (SCCS).

sccs applies the indicated *subcommand* to the history file associated with each of the indicated files.

The name of an SCCS history file is derived by prepending the '**s.**' prefix to the filename of a working copy. The **sccs** command normally expects these '**s.**files' to reside in an **SCCS** subdirectory. Thus, when you supply **sccs** with a *filename* argument, it normally applies the subcommand to a file named **s.***filename* in the **SCCS** subdirectory. If *filename* is a pathname, **sccs** looks for the history file in the **SCCS** subdirectory of that file's parent directory. If *filename* is a directory, however, **sccs** applies the subcommand to every **s.**file file it contains. Thus, the command:

```
example% sccs get program.c
```

would apply the **get** subcommand to a history file named:

```
SCCS/s.program.c
```

while the command:

```
example% sccs get SCCS
```

would apply it to every **s.**file in the **SCCS** subdirectory.

Options for the **sccs** command itself must appear before the *subcommand* argument. Options for a given subcommand must appear after the *subcommand* argument. These options are specific to each subcommand, and are described along with the subcommands themselves (see **Subcommands**, below).

Running Setuid

The **sccs** command also includes the capability to run ''setuid'' to provide additional protection. However this does not apply to subcommands such as **sccs-admin**(1), since this would allow anyone to change the authorizations of the history file. Commands that would do so always run as the real user.

OPTIONS

–r Run **sccs** with the real user ID, rather than set to the effective user ID.

–d*rootprefix*
Define the root portion of the pathname for SCCS history files. The default root portion is the current directory. Note: *rootprefix* is prepended to the entire *filename* argument, even if *filename* is an absolute pathname. **–d** overrides any directory specified by the **PROJECTDIR** environment variable (see ENVIRONMENT, below).

–p*subdir*
Define the (sub)directory within which a history file is expected to reside. **SCCS** is the default. (See EXAMPLES, below).

USAGE

Subcommands

Many of the following **sccs** subcommands invoke programs that reside in **/usr/ccs/bin**. Many of these subcommands accept additional arguments that are documented in the reference page for the utility program the subcommand invokes.

admin Modify the flags or checksum of an SCCS history file. Refer to **sccs-admin**(1) for more information about the **admin** utility. While **admin** can be used to initialize a history file, you may find that the **create** subcommand is simpler to use for this purpose.

cdc –**r***sid* [–**y**[*comment*]]

Annotate (change) the delta commentary. Refer to **sccs-cdc**(1). Note: The **fix** subcommand can be used to replace the delta, rather than merely annotating the existing commentary.

–**r***sid* Specify the SCCS delta ID (SID) to which the change notation is to be added. The SID for a given delta is a number, in Dewey decimal format, composed of two or four fields: the *release* and *level* fields, and for branch deltas, the *branch* and *sequence* fields. For instance, the SID for the initial delta is normally **1.1**.

–**y**[*comment*]

Specify the comment with which to annotate the delta commentary. If –**y** is omitted, **sccs** prompts for a comment. A null *comment* results in an empty annotation.

check [–**b**] [–**u**[*username*]]

Check for files currently being edited. Like **info** and **tell**, but returns an exit code, rather than producing a listing of files. **check** returns a non-zero exit status if anything is being edited.

–**b** Ignore branches.

–**u**[*username*]

Only check files being edited by you. When *username* is specified, only check files being edited by that user.

clean [–**b**]

Remove everything in the current directory that can be retrieved from an SCCS history. Does not remove files that are being edited.

–**b** Do not check branches to see if they are being edited. ‘**clean** –**b**’ is dangerous when branch versions are kept in the same directory.

comb Generate scripts to combine deltas. Refer to **sccs-comb**(1).

create Create (initialize) history files. **create** performs the following steps:

- Renames the original source file to **,program.c** in the current directory.
- Create the history file called **s.program.c** in the **SCCS** subdirectory.
- Performs an ‘**sccs get**’ on **program.c** to retrieve a read-only copy of the initial version.

deledit [–**s**] [–**y**[*comment*]]

Equivalent to an '**sccs delta**' and then an '**sccs edit**'. **deledit** checks in a delta, and checks the file back out again, but leaves the current working copy of the file intact.

–**s** Silent. Do not report delta numbers or statistics.

–**y**[*comment*]

Supply a comment for the delta commentary. If –**y** is omitted, **delta** prompts for a comment. A NULL *comment* results in an empty comment field for the delta.

delget [–**s**] [–**y**[*comment*]]

Perform an '**sccs delta**' and then an '**sccs get**' to check in a delta and retrieve read-only copies of the resulting new version. See the **deledit** subcommand for a description of –**s** and –**y**. **sccs** performs a **delta** on all the files specified in the argument list, and then a **get** on all the files. If an error occurs during the **delta**, the **get** is not performed.

delta [–**s**] **[**–y[*comment*]]

Check in pending changes. Records the line-by-line changes introduced while the file was checked out. The effective user ID must be the same as the ID of the person who has the file checked out. Refer to **sccs-delta**(1). See the **deledit** subcommand for a description of –**s** and –**y**.

diffs [–**C**] [–**c***date-time*] [–**r***sid*] *diff-options*

Compare (in **diff**(1) format) the working copy of a file that is checked out for editing, with a version from the SCCS history. Use the most recent checked-in version by default. The **diffs** subcommand accepts the same options as **diff**, with the exception that the –**c** option to **diff** must be specified as –**C**.

–**C** Pass the –**c** option to **diff**.

–**c***date-time*

Use the most recent version checked in before the indicated date and time for comparison. *date-time* takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]]. Omitted units default to their maximum possible values; that is –**c7502** is equivalent to –**c750228235959**.

–**r***sid* Use the version corresponding to the indicated delta for comparison.

edit Retrieve a version of the file for editing. '**sccs edit**' extracts a version of the file that is writable by you, and creates a **p.**file in the **SCCS** subdirectory as lock on the history, so that no one else can check that version in or out. ID keywords are retrieved in unexpanded form. **edit** accepts the same options as **get**, below.

enter Similar to **create**, but omits the final '**sccs get**'. This may be used if an '**sccs edit**' is to be performed immediately after the history file is initialized.

fix –**r***sid*

Revise a (leaf) delta. Remove the indicated delta from the SCCS history, but leave a working copy of the current version in the directory. This is useful for incorporating trivial updates for which no audit record is needed, or for revising the delta commentary. **fix** must be followed by a –**r** option, to specify the SID of the delta to remove. The indicated delta must be the most recent (leaf) delta in its branch. Use **fix** with caution since it does not leave an audit trail of differences (although the previous commentary is retained within the history file).

get [–**ekmps**] [–**c***date-time*] [–**r***sid*]

Retrieve a version from the SCCS history. By default, this is a read-only working copy of the most recent version; ID keywords are in expanded form. Refer to **sccs-get**(1).

–**e** Retrieve a version for editing. Same as **sccs edit**.

–**k** Retrieve a writable copy but do not check out the file. ID keywords are unexpanded.

–**m** Precede each line with the SID of the delta in which it was added.

–**p** Produce the retrieved version on the standard output. Reports that would normally go to the standard output (delta ID's and statistics) are directed to the standard error.

–**s** Silent. Do not report version numbers or statistics.

–**c***date-time*

Retrieve the latest version checked in prior to the date and time indicated by the *date-time* argument. *date-time* takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]].

–**r***sid* Retrieve the version corresponding to the indicated SID.

help *message-code* | *sccs-command*
help stuck

Supply more information about SCCS diagnostics. **help** displays a brief explanation of the error when you supply the code displayed by an SCCS diagnostic message. If you supply the name of an SCCS command, it prints a usage line. **help** also recognizes the keyword **stuck**. Refer to **sccs-help**(1).

info [–**b**] [–**u**[*username*]]

Display a list of files being edited, including the version number checked out, the version to be checked in, the name of the user who holds the lock, and the date and time the file was checked out.

–**b** Ignore branches.

–**u**[*username*]

Only list files checked out by you. When *username* is specified, only list files checked out by that user.

print Print the entire history of each named file. Equivalent to an '**sccs prs** –**e**' followed by an '**sccs get** –**p** –**m**'.

prs [–**el**] [–**c***date-time*] [–**r***sid*]

Peruse (display) the delta table, or other portion of an **s.**file. Refer to **sccs-prs**(1).

–**e**	Display delta table information for all deltas earlier than the one specified with –**r** (or all deltas if none is specified).
–**l**	Display information for all deltas later than, and including, that specified by –**c** or –**r**.
–**c***date-time*	Specify the latest delta checked in before the indicated date and time. The *date-time* argument takes the form: *yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]].
–**r***sid*	Specify a given delta by SID.

prt [–**y**] Display the delta table, but omit the MR field (see **sccsfile**(4) for more information on this field). Refer to **sccs-prt**(1).

–**y**	Display the most recent delta table entry. The format is a single output line for each filename argument, which is convenient for use in a pipeline with **awk**(1) or **sed**(1).

rmdel –**r***sid*

Remove the indicated delta from the history file. That delta must be the most recent (leaf) delta in its branch. Refer to **sccs-rmdel**(1).

sccsdiff –**r***old-sid* –**r***new-sid diff-options*

Compare two versions corresponding to the indicated SIDs (deltas) using **diff**. Refer to **sccs-sccsdiff**(1).

tell [–**b**] [–**u**[*username*]]

Display the list of files that are currently checked out, one filename per line.

–**b**	Ignore branches.
–**u**[*username*]	Only list files checked out to you. When *username* is specified, only list files check out to that user.

unedit “Undo” the last **edit** or ‘**get** –**e**’, and return the working copy to its previous condition. **unedit** backs out all pending changes made since the file was checked out.

unget Same as **unedit**. Refer to **sccs-unget**(1).

val Validate the history file. Refer to **sccs-val**(1).

what Display any expanded ID keyword strings contained in a binary (object) or text file. Refer to **what**(1) for more information.

EXAMPLES

sccs converts the command:

```
example% sccs –d/usr/src/include  get  stdio.h
```

to:

```
/usr/ccs/bin/get  /usr/src/include/SCCS/s.stdio.h
```

The command:

```
example% sccs –pprivate  get  include/stdio.h
```

becomes:

```
/usr/ccs/bin/get  include/private/s.stdio.h
```

To initialize the history file for a source file named **program.c**: make the SCCS subdirectory, and then use ‘**sccs create**’:

```
example% mkdir SCCS
example% sccs create program.c
program.c:
1.1
14 lines
```

After verifying the working copy, you can remove the backup file that starts with a comma:

```
example% diff program.c ,program.c
example% rm ,program.c
```

To check out a copy of **program.c** for editing, edit it, and then check it back in:

```
example% sccs edit program.c
1.1
new delta 1.2
14 lines
example% vi program.c
your editing session
example% sccs delget program.c
comments? clarified cryptic diagnostic
1.2
3 inserted
2 deleted
12 unchanged
1.2
15 lines
```

To retrieve a file from another directory into the current directory:

```
example% sccs get /usr/src/sccs/cc.c
```

or:

```
example% sccs –p/usr/src/sccs/ get cc.c
```

To check out all files under SCCS in the current directory:

```
example% sccs edit SCCS
```

To check in all files currently checked out to you:

```
example% sccs delta `sccs tell –u`
```

ENVIRONMENT

If the environment variable **PROJECTDIR** is set to contain an absolute pathname (beginning with a slash), **sccs** searches for SCCS history files in the directory given by that variable. If **PROJECTDIR** does not begin with a slash, it is taken as the name of a user, and **sccs** searches the **src** or **source** subdirectory of that user's home directory for history files.

FILES

SCCS	SCCS subdirectory
SCCS/d.*file*	temporary file of differences
SCCS/p.*file*	lock (permissions) file for checked-out versions
SCCS/q.*file*	temporary file
SCCS/s.*file*	SCCS history file
SCCS/x.*file*	temporary copy of the **s.**file
SCCS/z.*file*	temporary lock file
/usr/ccs/bin/∗	SCCS utility programs

SEE ALSO

awk(1), **diff**(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **sed**(1), **what**(1), **sccsfile**(4)

Programming Utilities Guide

BUGS

There is no **sact** subcommand to invoke **/usr/ccs/bin/sact** (see **sccs-sact**(1)). However, the **info** subcommand performs an equivalent function.

NAME

script – make record of a terminal session

SYNOPSIS

script [**–a**] [*filename*]

AVAILABILITY

SUNWcsu

DESCRIPTION

script makes a record of everything printed on your screen. The record is written to *filename*. If no file name is given, the record is saved in the file **typescript**.

The **script** command forks and creates a sub-shell, according to the value of **$SHELL**, and records the text from this session. The script ends when the forked shell exits or when CTRL-D is typed.

OPTIONS

–a	Append the session record to *filename*, rather than overwrite it.

NOTES

script places *everything* that appears on the screen in *filename*, including prompts.

NAME

sdiff – print differences between two files side-by-side

SYNOPSIS

sdiff [–**l**] [–**s**] [–**o** *output*] [–**w** *n*] *filename1 filename2*

AVAILABILITY

SUNWesu

DESCRIPTION

sdiff uses the output of the **diff** command to produce a side-by-side listing of two files indicating lines that are different. Lines of the two files are printed with a blank gutter between them if the lines are identical, a **<** in the gutter if the line appears only in *filename1*, a **>** in the gutter if the line appears only in *filename2*, and a **|** for lines that are different. (See the **EXAMPLES** section below.)

OPTIONS

–**l** Print only the left side of any lines that are identical.

–**s** Do not print identical lines.

–**o** *output*

Use the argument *output* as the name of a third file that is created as a user-controlled merge of *filename1* and *filename2*. Identical lines of *filename1* and *filename2* are copied to *output*. Sets of differences, as produced by **diff**, are printed; where a set of differences share a common gutter character. After printing each set of differences, **sdiff** prompts the user with a **%** and waits for one of the following user-typed commands:

l	Append the left column to the output file.
r	Append the right column to the output file.
s	Turn on silent mode; do not print identical lines.
v	Turn off silent mode.
e l	Call the editor with the left column.
e r	Call the editor with the right column.
e b	Call the editor with the concatenation of left and right.
e	Call the editor with a zero length file.
q	Exit from the program.

On exit from the editor, the resulting file is concatenated to the end of the *output* file.

–**w** *n* Use the argument *n* as the width of the output line. The default line length is 130 characters.

EXAMPLES

A sample output of **sdiff** follows.

```
x     |     y
a           a
b     <
c     <
d           d
      >     c
```

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **sdiff** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **sdiff** behaves.

LC_CTYPE

Determines how **sdiff** handles characters. When **LC_CTYPE** is set to a valid value, **sdiff** can display and handle text and filenames containing valid characters for that locale. **sdiff** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **sdiff** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

SEE ALSO

diff(1), **ed**(1), **environ**(5)

NAME

sed – stream editor

SYNOPSIS

sed [–**n**] [–**e** *script*] [–**f** *sfilename*] [*filename* . . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

sed copies the named *filename* (standard input default) to the standard output, edited according to a script of commands.

OPTIONS

–**n**	Suppress the default output.
–**e** *script*	*script* is an edit command for **sed**. See USAGE below for more information on the format of *script*. If there is just one –**e** option and no –**f** options, the flag –**e** may be omitted.
–**f** *sfilename*	Take the script from *sfilename*.

USAGE

A script consists of editing commands, one per line, of the following form:

[*address* **[,** *address* **]]** *function* **[** *arguments* **]**

In normal operation, **sed** cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and copies the resulting pattern space to the standard output (except under –**n**).

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, that is, a */regular expression/* in the style of **ed**(1) modified thus:

- In a context address, the construction \*?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note: In the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.
- The escape sequence **\n** matches a new-line *embedded* in the pattern space.
- A period **(.)** matches any character except the *terminal* new-line of the pattern space.
- A command line with no addresses selects every line.
- A command line with one address selects each line that matches the address.
- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second address. Thereafter the process is repeated, looking again for the first address. (If the second address is a number less than or equal to the line number selected by the first address, only the line corresponding to the first address is selected.)

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (described below).

In the following list of functions the maximum number of permissible addresses for each function is indicated.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

The following table lists the functions.

Maximum Number of Addresses	**Command**	**Description**
1	**a** *text*	Append. Place *text* on the output before reading the next input line.
2	**b** *label*	Branch to the **:** command bearing the *label*. If *label* is empty, branch to the end of the script.
2	**c** *text*	Change. Delete the pattern space. Place *text* on the output. Start the next cycle.
2	**d**	Delete the pattern space. Start the next cycle.
2	**D**	Delete the initial segment of the pattern space through the first new-line. Start the next cycle. (See the **N** command below.)
2	**g**	Replace the contents of the pattern space by the contents of the hold space.
2	**G**	Append the contents of the hold space to the pattern space.
2	**h**	Replace the contents of the hold space by the contents of the pattern space.
2	**H**	Append the contents of the pattern space to the hold space.
1	**i** *text*	Insert. Place *text* on the standard output.
2	**l**	List the pattern space on the standard output in an unambiguous form. Non-printable characters are displayed in octal notation and long lines are folded.
2	**n**	Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
2	**N**	Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)
2	**p**	Print. Copy the pattern space to the standard output.
2	**P**	Copy the initial segment of the pattern space through the first new-line to the standard output.
1	**q**	Quit. Branch to the end of the script. Do not start a new cycle.
2	**r** *rfile*	Read the contents of *rfile*. Place them on the output before reading the next input line.

Maximum Number of Addresses	**Com-mand**	**Description**
2	**t** *label*	Test. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.
2	**w** *wfile*	Write. Append the pattern space to *wfile*. The first occurrence of **w** will cause *wfile* to be cleared. Subsequent invocations of **w** will append. Each time the **sed** command is used, *wfile* is overwritten.
2	**x**	Exchange the contents of the pattern and hold spaces.
2	**!** *function*	Don't. Apply the *function* (or group, if *function* is **{**) only to lines *not* selected by the address(es).
0	**:** *label*	This command does nothing; it bears a *label* for **b** and **t** commands to branch to.
1	**=**	Place the current line number on the standard output as a line.
2	**{**	Execute the following commands through a matching **}** only when the pattern space is selected.
0		An empty command is ignored.
0	**#**	If a **#** appears as the first character on a line of a script file, then that entire line is treated as a comment, with one exception: if a **#** appears on the first line and the character after the **#** is an **n**, then the default output will be suppressed. The rest of the line after **#n** is also ignored. A script file must contain at least one non-comment line.

Maximum Number of Addresses	**Command** (Using *strings*) and **Description**
2	**s** / *regular expression* / *replacement* / *flags* Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of **/**. For a fuller description see **ed** (1). *flags* is zero or more of: *n n*= 1 - 512. Substitute for just the *n*th occurrence of the *regular expression.* **g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one. **p** Print the pattern space if a replacement was made. **w** *wfile* Write. Append the pattern space to *wfile* if a replacement was made.
2	**y** / *string1* / *string2* / Transform. Replace all occurrences of characters in *string1* with the corresponding characters in *string2*. *string1* and *string2* must have the same number of characters. For example, **y** / abc / ABC / replaces a with A, b with B, and c with C.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **sed** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **sed** behaves.

LC_CTYPE

Determines how **sed** handles characters. When **LC_CTYPE** is set to a valid value, **sed** can display and handle text and filenames containing valid characters for that locale. **sed** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **sed** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

awk(1), **ed**(1), **grep**(1), **environ**(5)

NAME

set, unset, setenv, unsetenv, export – shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents

SYNOPSIS

sh

set [**––aefhkntuvx** [*argument* ...]]

unset [*name* ...]

export [*name* ...]

csh

set [*var* [**=** *value*]]
set *var***[***n***]** **=** *word*

unset *pattern*

setenv [*VAR* [*word*]]

unsetenv *variable*

ksh

set [±**aefhkmnopstuvx**] [±**o** *option*]... [±**A** *name*] [*arg* ...]

unset [–**f**] *name* ...

†† **export** [*name*[**=***value*]] ...

DESCRIPTION

sh

The **set** built-in command has the following options:

–**a**	Mark variables which are modified or created for export.
–**e**	Exit immediately if a command exits with a non-zero exit status.
–**f**	Disable file name generation.
–**h**	Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
–**k**	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
–**n**	Read commands but do not execute them.
–**t**	Exit after reading and executing one command.
–**u**	Treat unset variables as an error when substituting.
–**v**	Print shell input lines as they are read.
–**x**	Print commands and their arguments as they are executed.
– –	Do not change any of the flags; useful in setting **$1** to –.

Using **+** rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**.
The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, **. . . .** If no arguments are given the values of all names are printed.

For each *name*, **unset** removes the corresponding variable or function value. The variables **PATH, PS1, PS2, MAILCHECK,** and **IFS** cannot be unset.

With the **export** built-in, the given *name*s are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

csh With no arguments, **set** displays the values of all shell variables. Multiword values are displayed as a parenthesized list. With the *var* argument alone, **set** assigns an empty (null) value to the variable *var*. With arguments of the form *var* **=** *value* **set** assigns *value* to *var*, where *value* is one of:

word	A single word (or quoted string).
(*wordlist***)**	A space-separated list of words enclosed in parentheses.

Values are command and filename expanded before being assigned. The form **set** *var***[***n***]** **=** *word* replaces the *n*'th word in a multiword value with *word*.

unset removes variables whose names match (filename substitution) *pattern*. All variables are removed by '**unset** ∗'; this has noticeably distasteful side effects.

With no arguments, **setenv** displays all environment variables. With the *VAR* argument, **setenv** sets the environment variable *VAR* to have an empty (null) value. (By convention, environment variables are normally given upper-case names.) With both *VAR* and *word* arguments, **setenv** sets the environment variable **NAME** to the value *word*, which must be either a single word or a quoted string. The most commonly used environment variables, **USER**, **TERM**, and **PATH**, are automatically imported to and exported from the **csh** variables **user**, **term**, and **path**; there is no need to use **setenv** for these. In addition, the shell sets the **PWD** environment variable from the **csh** variable **cwd** whenever the latter changes.

The environment variables **LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY** take immediate effect when changed within the C shell.

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **csh** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **csh** behaves.

LC_CTYPE

Determines how **csh** handles characters. When **LC_CTYPE** is set to a valid value, **csh** can display and handle text and filenames containing valid characters for that locale. **csh** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **csh** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S./English).

LC_NUMERIC
Determines the value of the radix character (decimal point (".") in the "C" locale) and thousand separator (empty string ("") in the "C" locale).

unsetenv removes *variable* from the environment. As with **unset**, pattern matching is not performed.

ksh The flags for the **set** built-in have meaning as follows:

–A Array assignment. Unset the variable *name* and assign values sequentially from the list *arg*. If **+A** is used, the variable *name* is not unset first.

–a All subsequent variables that are defined are automatically exported.

–e If a command has a non-zero exit status, execute the **ERR** trap, if set, and exit. This mode is disabled while reading profiles.

–f Disables file name generation.

–h Each command becomes a tracked alias when first encountered.

–k All variable assignment arguments are placed in the environment for a command, not just those that precede the command name.

–m Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.

–n Read commands and check them for syntax errors, but do not execute them. Ignored for interactive shells.

–o The following argument can be one of the following option names:

allexport
Same as –**a**.

errexit Same as –**e**.

bgnice All background jobs are run at a lower priority. This is the default mode. **emacs** Puts you in an **emacs** style in-line editor for command entry.

gmacs Puts you in a **gmacs** style in-line editor for command entry.

ignoreeof
The shell will not exit on end-of-file. The command **exit** must be used.

keyword
Same as –**k**.

markdirs

All directory names resulting from file name generation have a trailing **/** appended.

monitor	Same as –**m**.
noclobber	Prevents redirection **>** from truncating existing files. Require **>|** to truncate a file when turned on.
noexec	Same as –**n**.
noglob	Same as –**f**.
nolog	Do not save function definitions in history file.
nounset	Same as –**u**.
privileged	Same as –**p**.
verbose	Same as –**v**.
trackall	Same as –**h**.
vi	Puts you in insert mode of a **vi** style in-line editor until you hit escape character **033**. This puts you in control mode. A return sends the line.
viraw	Each character is processed as it is typed in **vi** mode.
xtrace	Same as –**x**.

If no option name is supplied then the current option settings are printed.

–**p**	Disables processing of the **$HOME/.profile** file and uses the file **/etc/suid_profile** instead of the **ENV** file. This mode is on whenever the effective uid is not equal to the real uid, or when the effective gid is not equal to the real gid. Turning this off causes the effective uid and gid to be set to the real uid and gid.
–**s**	Sort the positional parameters lexicographically.
–**t**	Exit after reading and executing one command.
–**u**	Treat unset parameters as an error when substituting.
–**v**	Print shell input lines as they are read.
–**x**	Print commands and their arguments as they are executed.
–	Turns off –**x** and –**v** flags and stops examining arguments for flags.
– –	Do not change any of the flags; useful in setting **$1** to a value beginning with –. If no arguments follow this flag then the positional parameters are unset.

Using **+** rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$**–. Unless –**A** is specified, the remaining arguments are positional parameters and are assigned, in order, to **$1 $2** . . .. If no arguments are given then the names and values of all variables are printed on the standard output.

The variables given by the list of *name*s are unassigned, i.e., their values and attributes are erased. **readonly** variables cannot be unset. If the –**f**, flag is set, then the names refer to *function* names. Unsetting **ERRNO**, **LINENO**, **MAILCHECK**, **OPTARG**, **OPTIND**, **RANDOM**,

SECONDS, **TMOUT**, and **_** removes their special meaning even if they are subsequently assigned.

When using **unset**, the variables given by the list of *name*s are unassigned, i.e., their values and attributes are erased. **readonly** variables cannot be unset. If the **−f**, flag is set, then the names refer to *function* names. Unsetting **ERRNO**, **LINENO**, **MAILCHECK**, **OPTARG**, **OPTIND**, **RANDOM**, **SECONDS**, **TMOUT**, and **_** removes their special meaning even if they are subsequently assigned.

With the **export** built-in, the given *name*s are marked for automatic export to the **environment** of subsequently-executed commands.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

csh(1), **ksh**(1), **read**(1), **sh**(1), **typeset**(1)

NAME

set, unset – set and unset local or global environment variables

SYNOPSIS

set [–**l** *variable*[**=***value*]] …
set [–**e** *variable*[**=***value*]] …
set [–**f***file variable*[**=***value*]] …

unset –**l** *variable* **…**
unset –**f***file variable* **…**

DESCRIPTION

The **set** command sets *variable* in the environment, or adds *variable***=***value* to *file*. If *variable* is not equated it to a value, **set** expects the value to be on *stdin*. The **unset** command removes *variable*. Note that the FMLI predefined, read-only variables (such as **ARG1**), may not be set or unset.

Note that at least one of the above options must be used for each variable being set or unset. If you set a variable with the –**f***filename* option, you must thereafter include *filename* in references to that variable. For example, **${(***file***)***VARIABLE***}**.

FMLI inherits the UNIX environment when invoked.

OPTIONS

–**l** Sets or unsets the specified variable in the local environment. Variables set with **-l** will not be inherited by processes invoked from FMLI.

–**e** Sets the specified variable in the UNIX environment. Variables set with **-e** will be inherited by any processes started from FMLI. Note that these variables cannot be **unset.**

-f*file* Sets or unsets the specified variable in the global environment. The argument *file* is the name, or pathname, of a file containing lines of the form *variable***=***value*. *file* will be created if it does not already exist. Note that no space intervenes between –**f** and *file*.

EXAMPLE

Storing a selection made in a menu:

```
name=Selection 2
action=`set –l SELECTION=2`close
```

NOTES

Variables set to be available to the UNIX environment (those set using the –**e** option) can only be set for the current fmli process and the processes it calls.

When using the –**f** option, unless *file* is unique to the process, other users of FMLI on the same machine will be able to expand these variables, depending on the read / write permissions on *file*.

A variable set in one frame may be referenced or unset in any other frame. This includes local variables.

SEE ALSO

env(1), **sh**(1)

NAME

setcolor – redefine or create a color

SYNOPSIS

setcolor *color red_level green_level blue_level*

DESCRIPTION

The **setcolor** command takes four arguments: *color*, which must be a string naming the color; and the arguments *red_level*, *green_level*, and *blue_level*, which must be integer values defining, respectively, the intensity of the red, green, and blue components of *color*. Intensities must be in the range of 0 to 1000. If you are redefining an existing color, you must use its current name (default color names are: **black**, **blue**, **green**, **cyan**, **red**, **magenta**, **yellow**, and **white**). **setcolor** returns the color's name string.

EXAMPLES

The following is an example of the arguments that **setcolor** takes:

```
`setcolor blue 100 24 300`
```

NAME

sh, jsh – shell: the standard shell, and job control shell -- command interpreters

SYNOPSIS

sh [**–acefhiknprstuvx**] [*argument*. . .]
jsh [**–acefhiknprstuvx**] [*argument*. . .]

AVAILABILITY

SUNWcsu

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. The command **jsh** is an interface to the shell which provides all of the functionality of **sh** and enables Job Control (see ''Job Control,'' below). See ''Invocation,'' below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of ASCII letters, digits, or underscores, beginning with a letter or an underscore. A *parameter* is a name, a digit, or any of the characters ∗, **@**, **#**, **?**, –, **$**, and **!\^**.

USAGE

Commands

A *simple-command* is a sequence of non-blank *word*s separated by *blank*s. The first *word* specifies the name of the command to be executed. Except as specified below, the remaining *word*s are passed as arguments to the invoked command. The command name is passed as argument 0 (see **exec**(2)). The *value* of a *simple-command* is its exit status if it terminates normally, or (octal) **200**+*status* if it terminates abnormally; see **signal**(5) for a list of status values.

A *pipeline* is a sequence of one or more *command*s separated by **|** . The standard output of each *command* but the last is connected by a **pipe**(2) to the standard input of the next *command.* Each *command* is run as a separate process; the shell waits for the last *command* to terminate. The exit status of a *pipeline* is the exit status of the last command in the *pipeline.*

A *list* is a sequence of one or more *pipeline*s separated by **;**, **&**, **&&**, or **| |** , and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and **| |** . The symbols **&&** and **| |** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding *pipeline* (that is, the shell waits for the *pipeline* to finish before executing any commands following the semicolon); an ampersand (**&**) causes asynchronous execution of the preceding pipeline (that is, the shell does *not* wait for that pipeline to finish). The symbol **&&** (**| |**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *simple-command* executed in the command.

for *name* [**in** *word* . . .] **do** *list* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* . . . is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see ‘‘Parameter Substitution,’’ below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] . . . **)** *list* **;;**] . . . **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see ‘‘File Name Generation’’) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list***; then** *list* **; [elif** *list* **; then** *list* **] . . . [else** *list* **] ; fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list***)**

Execute *list* in a sub-shell.

{ *list***;}**

list is executed in the current (that is, parent) shell. The **{** must be followed by a space.

name **() {** *list***;}**

Define a function which is referenced by *name*. The body of the function is the *list* of commands between **{** and **}**. The **{** must be followed by a space. Execution of functions is described below (see ‘‘Execution’’). The **{** and **}** are unnecessary if the body of the function is a *command* as defined above, under ‘‘Commands.’’

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments Lines

A word beginning with **#** causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The shell reads commands from the string between two grave accents (**‘ ‘**) and the standard output from these commands may be used as all or part of a word. Trailing newlines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (**‘**) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" **...‘...‘...** "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a newline character (**newline**), both the backslash and the newline are removed (see the later section on ‘‘Quoting’’). In addition, backslashes used to escape dollar signs (**$**) are removed. Since no parameter substitution is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, **‘**, ", **newline**, and **$** are left intact when the command string is read.

Parameter Substitution

The character **$** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] . . .

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter***}**

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is ∗ or **@**, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter***:–***word***}**

If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter***:=***word***}**

If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned in this way.

${*parameter***:?***word***}**

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message ‘‘parameter null or not set’’ is printed.

${*parameter***:+***word***}**

If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:–‘pwd‘}
```

If the colon (**:**) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell.

#	The number of positional parameters in decimal.
–	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell. The parameters in this section are also referred to as environment variables.

HOME	The default argument (home directory) for the **cd** command, set to the user's login directory by **login**(1) from the password file (see **passwd**(4)).
PATH	The search path for commands (see ‘‘Execution,’’ below).
CDPATH	The search path for the **cd** command.
MAIL	If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.
MAILCHECK	This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is **600** seconds (10 minutes). If set to **0**, the shell will check before each prompt.
MAILPATH	A colon (**:**) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by **%** and a message that will be printed when the modification time changes. The default message is **you have mail**.
PS1	Primary prompt string, by default ‘‘**$** ’’.
PS2	Secondary prompt string, by default ‘‘**>** ’’.
IFS	Internal field separators, normally **space**, **tab**, and **newline** (see ‘‘Blank Interpretation’’).

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed.

SHELL When the shell is invoked, it scans the environment (see ‘‘Environment,’’ below) for this name.

LC_CTYPE
Determines how the shell handles characters. When **LC_CTYPE** is set to a valid value, the shell can display and handle text and filenames containing valid characters for that locale. The shell can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. The shell can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

If **LC_CTYPE** and **LC_MESSAGES** (see **environ**(5)) are not set in the environment, the operational behavior of the shell for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how the shell behaves.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, and **IFS**. **HOME** and **MAIL** are set by **login**(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or **’’**) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Input/Output Redirection

A command’s input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are *not* passed on as arguments to the invoked command. Note: Parameter and command substitution occurs before *word* or *digit* is used.

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.

>>*word*	Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the EOF); otherwise, the file is created.
<<[–]*word*	After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an EOF. If, however, – is appended to **<<**: 1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*), 2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and 3) shell input is read up to the first line that literally matches the resulting *word*, or to an EOF. If any character of *word* is quoted (see ‘‘Quoting,’’ later), no additional processing is done to the shell input. If no characters of *word* are quoted: 1) parameter and command substitution occurs, 2) (escaped) **\newline**s are removed, and 3) \ must be used to quote the characters \, **$**, and **‘**. The resulting document becomes the standard input.
<&*digit*	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&***digit*.
<&–	The standard input is closed. Similarly for the standard output using **>&–**.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* **2>&1**

first associates file descriptor 1 with file *xxx* . It associates file descriptor 2 with the file associated with file descriptor 1 (that is, *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx* .

Using the terminology introduced on the first page, under ‘‘Commands,’’ if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by **&** the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

File Name Generation

Before a command is executed, each command *word* is scanned for the characters **∗**, **?**, and **[**. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character **.** at the start of a file name or immediately following a **/**, as well as the character **/** itself, must be matched explicitly.

∗	Matches any string, including the null string.
?	Matches any single character.
[. . .**]**	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening **[** is a **!**, any character not enclosed is matched.

Note: All quoted characters (see below) must be matched explicitly in a filename.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ˆ < > newline space tab

A character may be *quoted* (that is, made to stand for itself) by preceding it with a backslash (\ ) or inserting it between a pair of quote marks (**´´** or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **\newline** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (**´´**), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "**´**"), but a single quote can not be quoted inside a pair of single quotes.

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$∗** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("**$1 $2** . . ."); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("**$1**" "**$2**" . . .). \ quotes the characters \, **‘**, **,** and **$**. The pair **\newline** is removed before parameter and command substitution. If a backslash precedes characters other than \, **‘**, **,** **$**, and newline, then the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (that is, the value of **PS2**) is issued.

Environment

The *environment* (see **environ**(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set –a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 *command*
and
(export TERM; TERM=450; *command***)**

are equivalent as far as the execution of *command* is concerned if *command* is not a Special Command. If *command* is a Special Command, then

TERM=450 *command*

will modify the **TERM** variable in the current shell.

If the –**k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following example first prints **a=b c** and **c**:

```
echo a=b c
a=b c
set –k
echo a=b c
c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the command substitution, parameter substitution, blank interpretation, input/output redirection, and filename generation listed above are carried out. If the command name matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). If the command name does not match the name of a defined function, but matches one of the *Special Commands* listed below, it is executed in the shell process. The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via **exec**(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **/usr/bin**. The current directory is specified by a null path name, which can appear immediately after the equal sign, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a **/** the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash –r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location. When Job Control is enabled, additional *Special Commands* are added to the shell's environment (see ‘‘Job Control’’).

: No effect; the command does nothing. A zero exit code is returned.

. *filename*
Read and execute commands from *filename* and return. The search path specified by **PATH** is used to find the directory containing *filename*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*-th enclosing loop.

cd [*argument*]
Change the current directory to *argument*. The shell parameter **HOME** is the default *argument*. The shell parameter **CDPATH** defines the search path for the directory containing *argument*. Alternative directory names are separated by a colon (**:**). The default path is **<null>** (specifying the current directory). Note: The current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *argument* begins with a **/** the search path is not used. Otherwise, each directory in the path is searched for *argument*.

echo [*argument* ...]
Echo arguments. See **echo**(1) for usage and description.

eval [*argument* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*argument* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an EOF will also cause the shell to

exit.)

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

getopts
Use in shell scripts to support command syntax standards (see **intro**(1)); it parses positional parameters and checks for legal options. See **getoptcvt**(1) for usage and description.

hash [**−r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **−r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (∗) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*argument*]
Equivalent to **exec newgrp** *argument*. See **newgrp**(1M) for usage and description.

pwd Print the current working directory. See **pwd**(1) for usage and description.

read *name* ...
One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using **\newline**. Characters other than **newline** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is **0**, unless an EOF is encountered.

readonly [*name* ...]
The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**−−aefhkntuvx** [*argument* ...]]

–a	Mark variables which are modified or created for export.
–e	Exit immediately if a command exits with a non-zero exit status.
–f	Disable file name generation.
–h	Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
–k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
–n	Read commands but do not execute them.
–t	Exit after reading and executing one command.
–u	Treat unset variables as an error when substituting.
–v	Print shell input lines as they are read.
–x	Print commands and their arguments as they are executed.
– –	Do not change any of the flags; useful in setting **$1** to –.

Using **+** rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given the values of all names are printed.

shift [*n*]

The positional parameters from **$***n***+1** . . . are renamed **$1** . . . . If *n* is not given, it is assumed to be 1.

stop *pid* . . .

Halt execution of the process number *pid.* (see **ps**(1)).

test

Evaluate conditional expressions. See **test**(1) for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*argument*] [*n*] . . .

The command *argument* is to be read and executed when the shell receives numeric or symbolic signal(s) (*n*). (Note: *argument* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number or corresponding symbolic names. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *argument* is absent all trap(s) *n* are reset to their original values. If *argument* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *argument* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* . . .]

For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [–[**HS**][**a**| **cdfnstv**]]

ulimit [–[**HS**][**c**| **d**| **f**| **n**| **s**| **t**| **v**]] *limit*

ulimit prints or sets hard or soft resource limits. These limits are described in **getrlimit**(2).

If *limit* is not present, **ulimit** prints the specified limits. Any number of limits may be printed at one time. The –**a** option prints all limits.

If *limit* is present, **ulimit** sets the specified limit to *limit*. The string **unlimited** requests the largest valid limit. Limits may be set for only one resource at a time. Any user may set a soft limit to any value below the hard limit. Any user may lower a hard limit. Only a super-user may raise a hard limit; see **su**(1M).

The –**H** option specifies a hard limit. The –**S** option specifies a soft limit. If neither option is specified, **ulimit** will set both limits and print the soft limit.

The following options specify the resource whose limits are to be printed or set. If no option is specified, the file size limit is printed or set.

–**c**	maximum core file size (in 512-byte blocks)
–**d**	maximum size of data segment or heap (in kbytes)
–**f**	maximum file size (in 512-byte blocks)
–**n**	maximum file descriptor plus 1
–**s**	maximum size of stack segment (in kbytes)
–**t**	maximum CPU time (in seconds)
–**v**	maximum size of virtual memory (in kbytes)

umask [*nnn*]

The user file-creation mask is set to *nnn* (see **umask**(1)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]

For each *name*, remove the corresponding variable or function value. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK**, and **IFS** cannot be unset.

wait [*n*]

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through **exec**(2) and the first character of argument zero is –, commands are initially read from **/etc/profile** and from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/usr/bin/sh**. The flags below are interpreted by the shell on invocation only. Note: Unless the –**c** or –**s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c *string*	If the **−c** flag is present commands are read from *string*.
−i	If the **−i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.
−p	If the **−p** flag is present, the shell will not set the effective user and group IDs to the real user and group IDs.
−r	If the **−r** flag is present the shell is a restricted shell (see **rsh**(1M)).
−s	If the **−s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

The remaining flags and arguments are described under the **set** command above.

Job Control (jsh)

When the shell is invoked as **jsh**, Job Control is enabled in addition to all of the functionality described previously for **sh**. Typically Job Control is enabled for the interactive shell only. Non-interactive shells typically do not benefit from the added functionality of Job Control.

With Job Control enabled every command or pipeline the user enters at the terminal is called a *job*. All jobs exist in one of the following states: foreground, background or stopped. These terms are defined as follows: 1) a job in the foreground has read and write access to the controlling terminal; 2) a job in the background is denied read access and has conditional write access to the controlling terminal (see **stty**(1)); 3) a stopped job is a job that has been placed in a suspended state, usually as a result of a **SIGTSTP** signal (see **signal**(5)).

Every job that the shell starts is assigned a positive integer, called a *job number* which is tracked by the shell and will be used as an identifier to indicate a specific job. Additionally the shell keeps track of the *current* and *previous* jobs. The *current job* is the most recent job to be started or restarted. The *previous job* is the first non-current job.

The acceptable syntax for a Job Identifier is of the form:

%*jobid*

where, *jobid* may be specified in any of the following formats:

% or **+**	for the current job
−	for the previous job
?<*string*>	specify the job for which the command line uniquely contains *string.*
n	for job number *n*, where *n* is a job number
pref	where *pref* is a unique prefix of the command name (for example, if the command **ls −l name** were running in the background, it could be referred to as **%ls**); *pref* cannot contain blanks unless it is quoted.

When Job Control is enabled, the following commands are added to the user's environment to manipulate jobs:

bg [**%***jobid* . . .]
Resumes the execution of a stopped job in the background. If **%***jobid* is omitted the current job is assumed.

fg [**%***jobid* . . .]
Resumes the execution of a stopped job in the foreground, also moves an executing background job into the foreground. If **%***jobid* is omitted the current job is assumed.

jobs [**–p** | **–l**] [**%***jobid* ...]

jobs **–x** *command* [*arguments*]
Reports all jobs that are stopped or executing in the background. If **%***jobid* is omitted, all jobs that are stopped or running in the background will be reported. The following options will modify / enhance the output of **jobs**:

–l	Report the process group ID and working directory of the jobs.
–p	Report only the process group ID of the jobs.
–x	Replace any *jobid* found in *command* or *arguments* with the corresponding process group ID, and then execute *command* passing it *arguments.*

kill [**–signal**] **%***jobid*
Builtin version of **kill** to provide the functionality of the **kill** command for processes identified with a *jobid.*

stop **%***jobid* . . .
Stops the execution of a background job(s).

suspend
Stops the execution of the current shell (but not if it is the login shell).

wait [**%***jobid* . . .]
wait builtin accepts a job identifier. If **%***jobid* is omitted **wait** behaves as described above under **Special Commands**.

EXIT CODES

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

jsh Only

If the shell is invoked as **jsh** and an attempt is made to exit the shell while there are stopped jobs, the shell issues one warning:

There are stopped jobs.

This is the only message. If another exit attempt is made, and there are still stopped jobs they will be sent a **SIGHUP** signal from the kernel and the shell is exited.

FILES

$HOME/.profile
/dev/null
/etc/profile
/tmp/sh*

SEE ALSO

intro(1), **echo**(1), **getoptcvt**(1) **login**(1), **pwd**(1), **ps**(1), **shell_builtins**(1), **stty**(1), **rsh**(1M), **newgrp**(1M), **dup**(2), **exec**(2), **fork**(2), **getrlimit**(2), **pipe**(2), **ulimit**(2), **setlocale**(3C), **profile**(4), **passwd**(4), **environ**(5), **signal**(5)

NOTES

Words used for filenames in input/output redirection are not interpreted for filename generation (see ‘‘File Name Generation,’’ above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message *cannot fork, too many processes*, try using the **wait**(1) command to clean up your background processes. If this doesn’t help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

Only the last process in a pipeline can be waited for.

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to **exec** the original command. Use the **hash** command to correct this situation.

NAME

shell – run a command using shell

SYNOPSIS

shell *command* [*command*] . . .

DESCRIPTION

The **shell** function concatenate its arguments, separating each by a space, and passes this string to the shell (**$SHELL** if set, otherwise **/usr/bin/sh**).

EXAMPLES

Since the Form and Menu Language does not directly support background processing, the **shell** function can be used instead.

```
`shell "build prog > /dev/null &"`
```

If you want the user to continue to be able to interact with the application while the background job is running, the output of an executable run by **shell** in the background must be redirected: to a file if you want to save the output, or to **/dev/null** if you don't want to save it (or if there is no output), otherwise your application may appear to be hung until the background job finishes processing.

shell can also be used to execute a command that has the same name as an FMLI built-in function.

NOTES

The arguments to **shell** will be concatenate using spaces, which may or may not do what is expected. The variables set in local environments will not be expanded by the shell because "local" means "local to the current process."

SEE ALSO

sh(1)

NAME

shell_builtins – shell command interpreter built-in functions

DESCRIPTION

The shell command interpreters (**sh**(1), **csh**(1), and **ksh**(1)), have special built-in functions which are interpreted by the shell as commands. Many of these built-in commands are implemented by more than one of the shells, and some are unique to a particular shell. These are:

alias
bg
break
case
cd
chdir
continue
dirs
eval
exec
exit
export
fc
fg
for
foreach
function
getopts
glob
goto
hash
hashstat
history
if
jobs
kill
let
limit
logout
newgrp
notify
onintr
popd
print
pushd
read
readonly
rehash
repeat
return
select
set
setenv
shift
source
stop
suspend
switch
test
times
trap
type
typeset
ulimit
umask
unalias
unhash
unlimit
unset
unsetenv
until
wait
whence
while

Bourne Shell, sh, Special Commands

Input / output redirection is now permitted for these commands. File descriptor 1 is the default output location. When Job Control is enabled, additional *Special Commands* are added to the shell's environment.

Additional to these built-in reserved command words, **sh** also uses:

: No effect; the command does nothing. A zero exit code is returned.

. *filename*
Read and execute commands from *filename* and return. The search path specified by **PATH** is used to find the directory containing *filename*.

C shell, csh

Built-in commands are executed within the C shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell. Additional to these built-in reserved command words, **csh** also uses:

: Null command. This command is interpreted, but performs no action.

Korn Shell, ksh, Special Commands

Input / Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is zero.

Commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.

2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

Additional to these built-in reserved command words, **ksh** also uses:

† **:** [*arg* . . .]
The command only expands parameters.

† **.** *file* **[** *arg* **. . .]**
Read the complete *file* then execute the commands. The commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed. the loop termination test.

SEE ALSO

alias(1), **break**(1), **case**(1), **cd**(1), **chmod**(1), **csh**(1), **echo**(1), **exec**(1), **exit**(1), **for**(1), **find**(1), **function**(1), **getoptcvt**(1) **getopts**(1), **glob**(1), **hash**(1), **history**(1), **if**(1), **intro**(1), **jobs**(1), **kill**(1), **ksh**(1), **let**(1), **limit**(1), **login**(1), **logout**(1), **newgrp**(1), **nice**(1), **nohup**(1), **print**(1), **pwd**(1), **read**(1), **readonly**(1), **repeat**(1), **set**(1), **shift**(1), **sh**(1), **suspend**(1), **time**(1), **times**(1), **trap**(1), **typeset**(1), **umask**(1), **wait**(1), **while**(1), **test**(1B), **chdir**(2), **chmod**(2), **creat**(2), **umask**(2), **getopt**(3C), **profile**(4), **environ**(5)

NAME

shift – shell built-in function to traverse either a shell's argument list or a list of field-separated words

SYNOPSIS

sh

shift [*n*]

csh

shift [*variable*]

ksh

† **shift** [*n*]

DESCRIPTION

sh

The positional parameters from **$***n***+1** . . . are renamed **$1** . . . . If *n* is not given, it is assumed to be 1.

csh

The components of **argv**, or *variable*, if supplied, are shifted to the left, discarding the first component. It is an error for the variable not to be set or to have a null value.

ksh

The positional parameters from **$***n***+1 $***n***+1 . . .** are renamed **$1 . . .**, default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

csh(1), **ksh**(1), **sh**(1)

NAME

shutdown – close down the system at a given time

SYNOPSIS

/usr/ucb/shutdown [**–fhknr**] *time* [*warning-message* ...]

AVAILABILITY

SUNWscpu

DESCRIPTION

shutdown provides an automated procedure to notify users when the system is to be shut down. *time* specifies when **shutdown** will bring the system down; it may be the word **now** (indicating an immediate shutdown), or it may specify a future time in one of two formats: **+***number* and *hour***:***min*. The first form brings the system down in *number* minutes, and the second brings the system down at the time of day indicated in 24-hour notation.

At intervals that get closer as the apocalypse approaches, warning messages are displayed at terminals of all logged-in users, and of users who have remote mounts on that machine.

At shutdown time a message is written to the system log daemon, **syslogd**(1M), containing the time of shutdown, the instigator of the shutdown, and the reason. Then a terminate signal is sent to **init**, which brings the system down to single-user mode.

OPTIONS

As an alternative to the above procedure, these options can be specified:

–f	Arrange, in the manner of **fastboot**(1B), that when the system is rebooted, the file systems will not be checked.
–h	Execute **halt**(1M).
–k	Simulate shutdown of the system. Do not actually shut down the system.
–n	Prevent the normal **sync**(2) before stopping.
–r	Execute **reboot**(1M).

FILES

/etc/xtab	list of remote hosts that have mounted this host

SEE ALSO

login(1), **fastboot**(1B), **halt**(1M), **reboot**(1M), **syslogd**(1M), **sync**(2)

NOTES

Only allows you to bring the system down between **now** and 23:59 if you use the absolute time for shutdown.

NAME

size – print section sizes in bytes of object files

SYNOPSIS

size [**–f**] [**–F**] [**–n**] [**–o**] [**–V**] [**–x**] *filename* . . .

DESCRIPTION

The **size** command produces segment or section size information in bytes for each loaded section in ELF or COFF object files. **size** prints out the size of the text, data, and bss (uninitialized data) segments (or sections) and their total.

size processes ELF and COFF object files entered on the command line. If an archive file is input to the **size** command, the information for each object file in the archive is displayed.

When calculating segment information, the **size** command prints out the total file size of the non-writable segments, the total file size of the writable segments, and the total memory size of the writable segments minus the total file size of the writable segments.

If it cannot calculate segment information, **size** calculates section information. When calculating section information, it prints out the total size of sections that are allocatable, non-writable, and not NOBITS, the total size of the sections that are allocatable, writable, and not NOBITS, and the total size of the writable sections of type NOBITS. (NOBITS sections do not actually take up space in the *filename*.)

If **size** cannot calculate either segment or section information, it prints an error message and stops processing the file.

OPTIONS

–f Print out the size of each allocatable section, the name of the section, and the total of the section sizes. If there is no section data, **size** prints out an error message and stops processing the file.

–F Print out the size of each loadable segment, the permission flags of the segment, then the total of the loadable segment sizes. If there is no segment data, **size** prints an error message and stops processing the file.

–n Print out non-loadable segment or non-allocatable section sizes. If segment data exists, **size** prints out the memory size of each loadable segment or file size of each non-loadable segment, the permission flags, and the total size of the segments. If there is no segment data, **size** prints out, for each allocatable and non-allocatable section, the memory size, the section name, and the total size of the sections. If there is no segment or section data, **size** prints an error message and stops processing.

–o Print numbers in octal, not decimal.

–V Print the version information for the **size** command on the standard error output.

–x Print numbers in hexadecimal; not decimal.

EXAMPLES

The examples below are typical **size** output.

```
example% size filename
2724 + 88 + 0 = 2812

example% size –f filename
26(.text) + 5(.init) + 5(.fini) = 36

example% size –F filename
2724(r-x) + 88(rwx) + 0(rwx) = 2812     (If statically linked)
```

SEE ALSO

as(1), **cc**(1B), **ld**(1), **a.out**(4), **ar**(4)

NOTES

Since the size of bss sections is not known until link-edit time, the **size** command will not give the true total size of pre-linked objects.

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep *time*

AVAILABILITY

SUNWcsu

DESCRIPTION

sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

ENVIRONMENT

If any of the **LC_***∗* variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **sleep** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **sleep** behaves.

LC_CTYPE

Determines how **sleep** handles characters. When **LC_CTYPE** is set to a valid value, **sleep** can display and handle text and filenames containing valid characters for that locale. **sleep** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **sleep** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

alarm(2), **sleep**(3C), **environ**(5)

NAME

soelim – resolve and eliminate .so requests from nroff or troff input

SYNOPSIS

soelim [*filename* . . .]

AVAILABILITY

SUNWdoc

DESCRIPTION

soelim reads the specified files or the standard input and performs the textual inclusion implied by the **nroff**(1) directives of the form

.so *somefile*

when they appear at the beginning of input lines. This is useful since programs such as **tbl**(1) do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of '–' is taken to be a file name corresponding to the standard input.

Note: Inclusion can be suppressed by using '´' instead of '**.**', that is,

´ so /usr/share/lib/tmac/tmac.s

EXAMPLES

A sample usage of **soelim** would be

```
example% soelim exum?.n | tbl | nroff −ms | col | lpr
```

SEE ALSO

more(1), **nroff**(1), **tbl**(1)

NAME

sort – sort and / or merge files

SYNOPSIS

sort [**–cmu**] [**–o***output*] [**–T** *directory*] [**–y***kmem*] [**–dfiMnr**] [**–bt***x*]
[**+***pos1* [**–***pos2*]] [*filename . . .*]

AVAILABILITY

SUNWdoc

DESCRIPTION

The **sort** command sorts lines of all the named files together and writes the result on the standard output. The standard input is read if '–' is used as a file name or no input-files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

OPTIONS

The following options alter the default behavior:

–c	Check that the input-file is sorted according to the ordering rules; give no output unless the file is out of sort.
–m	Merge only, the input-files are already sorted.
–u	Unique: suppress all but one in each set of lines having equal keys.
–o*output*	The argument given is the name of an output-file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **–o** and *output.*
–T *directory*	The *directory* argument is the name of a directory in which to place temporary files.
–y*kmem*	The amount of main memory used by **sort** has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, **sort** begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value *kmem*, **sort** will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **–y**0 is guaranteed to start with minimum memory. By convention, **–y** (with no argument) starts with maximum memory.

The following options override the default ordering rules:

–d	‘‘Dictionary’’ order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.
–f	Fold lower-case letters into upper case.
–i	Ignore non-printable characters.

–M	Compare as months. The first three non-blank characters of the field are folded to upper case and compared. For example, in English the sorting order is "JAN" < "FEB" < . . . < "DEC". Invalid fields compare low to "JAN". The **–M** option implies the **–b** option (see below).
–n	An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **–n** option implies the **–b** option (see below). Note: The **–b** option is only effective when restricted sort key specifications are in effect.
–r	Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation **+***pos1* *–pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at position *pos1* and just before *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *–pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

–b	Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **–b** option is specified before the first **+***pos1* argument, it will be applied to all **+***pos1* arguments. Otherwise, the **b** flag may be attached independently to each **+***pos1* or *–pos2* argument (see below).
–t*x*	Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

pos1 and *pos2* each have the form *m***.***n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by **+***m***.***n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing **.***n* means **.**0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; **+***m***.**0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *–m***.***n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing **.***n* means **.**0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *–m***.**1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *input-file* with the second field as the sort key:

```
example% sort +1 –2 input-file
```

Sort, in reverse order, the contents of *input-file1* and *input-file2*, placing the output in *output-file* and using the first character of the second field as the sort key:

```
example% sort –r –o output-file +1.0 –1.2 input-file1 input-file2
```

Sort, in reverse order, the contents of *input-file1* and *input-file2* using the first non-blank character of the second field as the sort key:

```
example% sort –r +1.0b –1.1b input-file1 input-file2
```

Print the password file, **passwd**(4), sorted by the numeric user ID (the third colon-separated field):

```
example% sort –t: +2n –3 /etc/passwd
```

Print the lines of the already sorted file *input-file*, suppressing all but the first occurrence of lines having the same third field (the options **–um** with just one input-file make the choice of a unique representative from a set of equal lines predictable):

```
example% sort –um +2 –3 input-file
```

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **sort** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **sort** behaves.

LC_CTYPE

Determines how **sort** handles characters. When **LC_CTYPE** is set to a valid value, **sort** can display and handle text and filenames containing valid characters for that locale. **sort** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **sort** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME

Determines how **sort** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/var/tmp/stm???	temporary files

SEE ALSO

comm(1), **join**(1), **uniq**(1), **environ**(5)

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorders discovered under the **–c** option.

NOTES

When the last line of an input-file is missing a **new-line** character, **sort** appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.

NAME

sortbib – sort a bibliographic database

SYNOPSIS

sortbib [**–s***KEYS*] *database*. . .

AVAILABILITY

SUNWdoc

DESCRIPTION

sortbib sorts files of records containing **refer** key-letters by user-specified keys. Records may be separated by blank lines, or by '**.[**' and '**.]**' delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so **sortbib** may not be used in a pipeline to read standard input.

The most common key-letters and their meanings are given below.

%A	Author's name
%B	Book containing article referenced
%C	City (place of publication)
%D	Date of publication
%E	Editor of book containing article referenced
%F	Footnote number or label (supplied by **refer**)
%G	Government order number
%H	Header commentary, printed before reference
%I	Issuer (publisher)
%J	Journal containing article
%K	Keywords to use in locating reference
%L	Label field used by –**k** option of **refer**
%M	Bell Labs Memorandum (undefined)
%N	Number within volume
%O	Other commentary, printed at end of reference
%P	Page number(s)
%Q	Corporate or Foreign Author (unreversed)
%R	Report, paper, or thesis (unpublished)
%S	Series title
%T	Title of article or book
%V	Volume number
%X	Abstract — used by **roffbib**, not by **refer**
%Y,Z	Ignored by **refer**

By default, **sortbib** alphabetizes by the first **%A** and the **%D** fields, which contain the senior author and date.

sortbib sorts on the last word on the **%A** line, which is assumed to be the author's last name. A word in the final position, such as '**jr.**' or '**ed.**', will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the **nroff** convention '**\0**' in place of a blank. A **%Q** field is considered to be the same as **%A**, except sorting begins with the first, not the last, word. **sortbib** sorts on the last word of the **%D** line, usually the year. It also ignores leading articles (like '**A**' or '**The**') when sorting by titles in the **%T** or **%J** fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, **sortbib** places that record before other records containing that field.

No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

OPTIONS

–**s***KEYS* Specify new *KEYS*. For instance, –**sATD** will sort by author, title, and date, while –**sA+D** will sort by all authors, and date. Sort keys past the fourth are not meaningful.

SEE ALSO

addbib(1), **indxbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1)

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell – find spelling errors

SYNOPSIS

spell [–**bvx**] [**+***local_file*] [*filename*] . . .

AVAILABILITY

SUNWesu

DESCRIPTION

spell collects words from the named *filename*s and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *filename*s are named, words are collected from the standard input. Copies of all output are accumulated in the **spellhist** file.

OPTIONS

–**b**	British spelling is checked. Besides preferring "centre," "colour," "programme," "speciality," "travelled," and so forth, this option insists upon *–ise* in words like "standard*ise*."
–**v**	All words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.
–**x**	Every plausible stem is displayed, one per line, with **=** preceding each word.
+*local_file*	*local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to **spell**'s own spelling list) for each job. Words found in *local_file* are removed from **spell**'s output. Use **sort**(1) to order *local_file* in ASCII collating sequence. If this ordering is not followed, some entries in *local_file* may be ignored.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/var/adm/spellhist	history file
/usr/share/lib/dict/words	master dictionary

SEE ALSO

sort(1)

NOTES

Because copies of all output are accumulated in the **spellhist** file, **spellhist** may grow quite large and require purging.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [**–aknpx**] . . .

AVAILABILITY

SUNWesu

DESCRIPTION

spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by **graph**(1).

OPTIONS

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be **1** if next argument is not a number.

–k The constant *k* used in the boundary value computation

$$y''_0 = ky''_1, \quad y''_n = ky''_{n-1}101$$

is set by the next argument. By default $k = 0$.

–n Space output points so that approximately *n* intervals occur between the lower and upper *x* limits. (Default $n = 100$.)

–p Make output periodic, that is, match derivatives at ends. First and last input values should normally agree.

–x Next 1 (or 2) arguments are lower (and upper) *x* limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1)

R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed.

DIAGNOSTICS

When data is not strictly monotonic in *x*, **spline** reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

NAME

split – split a file into pieces

SYNOPSIS

split [–*n*] [*filename* [*name*]]

AVAILABILITY

SUNWesu

DESCRIPTION

split reads *filename* and writes it in *n*-line pieces into a set of output-files. The name of the first output-file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). The maximum length of *name* is 2 characters less than the maximum filename length allowed by the filesystem. See **statvfs**(2). If no output name is given, **x** is used as the default (output-files will be called **xaa**, **xab**, and so forth).

If no input-file is given, or if – is given in its stead, then the standard input-file is used.

OPTIONS

–*n*	Number of lines in each piece. Defaults to 1000 lines.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **split** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **split** behaves.

LC_CTYPE

Determines how **split** handles characters. When **LC_CTYPE** is set to a valid value, **split** can display and handle text and filenames containing valid characters for that locale. **split** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **split** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

csplit(1), **statvfs**(2), **environ**(5)

NAME

srchtxt – display contents of, or search for a text string in, message data bases

SYNOPSIS

srchtxt [–**s**] [–**l** *locale*] [–**m** *msgfile*, . . .] [*text*]

AVAILABILITY

SUNWloc

DESCRIPTION

The **srchtxt** utility is used to display all the text strings in message data bases, or to search for a text string in message data bases (see **mkmsgs**(1)). These data bases are files in the directory **/usr/lib/locale/***locale***/LC_MESSAGES** (see **setlocale**(3C)), unless a file name given with the –**m** option contains a **/**. The directory *locale* can be viewed as the name of the language in which the text strings are written. If the –**l** option is not specified, the files accessed will be determined by the value of the environment variable **LC_MESSAGES**. If **LC_MESSAGES** is not set, the files accessed will be determined by the value of the environment variable **LANG**. If **LANG** is not set, the files accessed will be in the directory **/usr/lib/locale/C/LC_MESSAGES**, which contains default strings.

If no *text* argument is present, then all the text strings in the files accessed will be displayed.

If the –**s** option is not specified, the displayed text is prefixed by message sequence numbers. The message sequence numbers are enclosed in angle brackets: **<***msgfile***:***msgnum***>.**

msgfile	name of the file where the displayed text occurred
msgnum	sequence number in *msgfile* where the displayed text occurred

This display is in the format used by **gettxt**(1) and **gettxt**(3C).

OPTIONS

–**s**	Suppress printing of the message sequence numbers of the messages being displayed.
–**l** *locale*	Access files in the directory **/usr/lib/locale/***locale***/LC_MESSAGES**. If –**m** *msgfile* is also supplied, *locale* is ignored for *msgfile*s containing a **/**.
–**m** *msgfile*	Access files specified by one or more *msgfile*s. If *msgfile* contains a **/** character, then *msgfile is* interpreted as a pathname; otherwise, it will be assumed to be in the directory determined as described above. To specify more than one *msgfile*, separate the file names using commas.
text	Search for the text string specified by *text* and display each one that matches. *text* can take the form of a regular expression (see **ed**(1)).

EXAMPLES

The following examples show uses of **srchtxt**.

Example 1:

If message files have been installed in a locale named **french** by using **mkmsgs**(1), then you could display the entire set of text strings in the **french** locale (**/usr/lib/locale/french/LC_MESSAGES/**∗) by typing:

```
example% srchtxt –l french
```

Example 2:

If a set of error messages associated with the operating system have been installed in the file **UX** in the **french** locale (**/usr/lib/locale/french/LC_MESSAGES/UX**), then, using the value of the **LANG** environment variable to determine the locale to be searched, you could search that file in that locale for all error messages dealing with files by typing:

```
example% setenv LANG=french; export LANG
example% srchtxt –m UX "[Ff]ichier"
```

If **/usr/lib/locale/french/LC_MESSAGES/UX** contained the following strings:

```
Erreur E/S \n
Liste d’arguments trop longue \n
Fichier inexistant \n
Argument invalide \n
Trop de fichiers ouverts \n
Fichier trop long \n
Trop de liens \n
Argument hors du domaine \n
Identificateur supprim \n
Etreinte fatale \n
  .
  .
  .
```

then the following strings would be displayed:

```
<UX:3>Fichier inexistant \n
<UX:5>Trop de fichiers ouverts \n
<UX:6>Fichier trop long \n
```

Example 3:

If a set of error messages associated with the operating system have been installed in the file **UX** and a set of error messages associated with the INGRESS data base product have been installed in the file **ingress**, both in the **german** locale, then you could search for the pattern **[Dd]atei** in both the files **UX** and **ingress** in the **german** locale by typing:

```
example% srchtxt –l german –m UX,ingress "[Dd]atei"
```

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **srchtxt** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **srchtxt** behaves.

LC_CTYPE Determines how **srchtxt** handles characters. When **LC_CTYPE** is set to a valid value, **srchtxt** can display and handle text and filenames containing valid characters for that locale. **srchtxt** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **srchtxt** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

FILES

/usr/lib/locale/C/LC_MESSAGES/*	default files created by **mkmsgs**(1)
/usr/lib/locale/*locale***/LC_MESSAGES/***	message files created by **mkmsgs**(1)

SEE ALSO

ed(1), **exstr**(1), **gettxt**(1), **mkmsgs**(1), **gettxt**(3C), **setlocale**(3C), **environ**(5)

DIAGNOSTICS

The error messages produced by **srchtxt** are intended to be self-explanatory. They indicate an error in the command line or errors encountered while searching for a particular locale and / or message file.

NAME

strchg, strconf – change or query stream configuration

SYNOPSIS

strchg –**h** *module1*[,*module2*. . .]
strchg –**p** [–**a** | –**u** *module*]
strchg –**f** *filename*

strconf [–**m** | –**t** *module*]

AVAILABILITY

SUNWcsu

DESCRIPTION

These commands are used to alter or query the configuration of the stream associated with the user's standard input. The **strchg** command pushes modules on and / or pops modules off the stream. The **strconf** command queries the configuration of the stream. Only the super-user or owner of a STREAMS device may alter the configuration of that stream.

Invoked without any arguments, **strconf** prints a list of all the modules in the stream as well as the topmost driver. The list is printed with one name per line where the first name printed is the topmost module on the stream (if one exists) and the last item printed is the name of the driver.

OPTIONS

The following options apply to **strchg** and, –**h,** –**f,** and –**p** are mutually exclusive.

–**h** *module1* [, *module2*. . .]
Mnemonic for pus*h*, pushes modules onto a stream. It takes as arguments the names of one or more pushable streams modules. These modules are pushed in order; that is, *module1* is pushed first, *module2* is pushed second, etc.

–**p** Mnemonic for po*p*, pops modules off the stream. With the –**p** option alone, **strchg** pops the topmost module from the stream.

–**a** *module* Pop all the modules above the topmost driver off the stream. This option requires the –**p** option.

–**u** *module* All modules above, but not including *module* are popped off the stream. This option requires the –**p** option.

–**f** *filename* Specify a *filename* that contains a list of modules representing the desired configuration of the stream. Each module name must appear on a separate line where the first name represents the topmost module and the last name represents the module that should be closest to the driver. **strchg** will determine the current configuration of the stream and pop and push the necessary modules in order to end up with the desired configuration.

The following options apply to **strconf** and, –**m** and –**t** are mutually exclusive.

–**m** *module*
Determine if the named *module* is present on a stream. If it is, **strconf** prints the message **yes** and returns zero. If not, **strconf** prints the message **no** and returns a non-zero value. The –**t** and –**m** options are mutually exclusive.

–t *module*

Print only the topmost module (if one exists). The **–t** and **–m** options are mutually exclusive.

EXAMPLES

The following command pushes the module **ldterm** on the stream associated with the user's standard input:

```
example% strchg –h ldterm
```

The following command pops the topmost module from the stream associated with **/dev/term/24**. The user must be the owner of this device or the super-user.

```
example% strchg –p < /dev/term/24
```

If the file **fileconf** contains the following:

```
ttcompat
ldterm
ptem
```

then the command

```
example% strchg –f fileconf
```

will configure the user's standard input stream so that the module **ptem** is pushed over the driver, followed by **ldterm** and **ttcompat** closest to the stream head.

The **strconf** command with no arguments lists the modules and topmost driver on the stream; for a stream that has only the module **ldterm** pushed above the **zs** driver, it would produce the following output:

```
ldterm
zs
```

The following command asks if **ldterm** is on the stream

```
example% strconf –m ldterm
```

and produces the following output while returning an exit status of 0:

```
yes
```

SEE ALSO

streamio(7)

DIAGNOSTICS

strchg returns zero on success. It prints an error message and returns non-zero status for various error conditions, including usage error, bad module name, too many modules to push, failure of an ioctl on the stream, or failure to open *filename* from the **–f** option.

strconf returns zero on success (for the **–m** or **–t** option, "success" means the named or topmost module is present). It returns a non-zero status if invoked with the **–m** or **–t** option and the module is not present. It prints an error message and returns non-zero status for various error conditions, including usage error or failure of an **ioctl** on the stream.

NOTES

If the user is neither the owner of the stream nor the super-user, the **strchg** command will fail. If the user does not have read permissions on the stream and is not the super-user, the **strconf** command will fail.

If modules are pushed in the wrong order, one could end up with a stream that does not function as expected. For ttys, if the line discipline module is not pushed in the correct place, one could have a terminal that does not respond to any commands.

NAME

strings – find printable strings in an object or binary file

SYNOPSIS

strings [–**a**] [–**o**] [–*number*] [*filename*. . .]

DESCRIPTION

The **strings** command looks for ASCII strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null character.

strings is useful for identifying random object files and many other things.

OPTIONS

–**a**	Look everywhere in the file for strings. If this flag is omitted, **strings** only looks in the initialized data space of object files.
–**o**	Precede each string by its offset in the file.
–*number*	Use a *number* as the minimum string length rather than the default, which is *4*.

ENVIRONMENT

LC_CTYPE determines how **strings** handles characters. When **LC_CTYPE** is set to a valid value, **strings** can display and handle text and filenames containing valid characters for that locale. **strings** can display and handle Extended Unix Code (EUC) characters where any character can be 1, 2, or 3 bytes wide. **strings** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

SEE ALSO

od(1)

NOTES

The algorithm for identifying strings is extremely primitive.

For backwards compatibility, the options –**a** or just the **-** are interchangeable.

NAME

strip – strip symbol table, debugging and line number information from an object file

SYNOPSIS

strip [–**blrVx**] *filename*. . .

DESCRIPTION

The **strip** command strips the symbol table, debugging information, and line number information from **ELF** object files. Once this stripping process has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

If **strip** is executed on a common archive file (see **ar**(4)) in addition to processing the members, **strip** will remove the archive symbol table. The archive symbol table must be restored by executing the **ar**(1) command with the –**s** option before the archive can be linked by the **ld**(1) command. **strip** will produce appropriate warning messages when this situation arises.

strip is used to reduce the file storage overhead taken by the object file.

OPTIONS

The amount of information stripped from the **ELF** object file can be controlled by using any of the following options:

–**b**	Same effect as the default behavior. This option is obsolete and will be removed in the next release.
–**l**	Strip line number information only; do not strip the symbol table or debugging information.
–**r**	Same effect as the default behavior. This option is obsolete and will be removed in the next release.
–**V**	Print, on standard error, the version number of **strip**.
–**x**	Do not strip the symbol table; debugging and line number information may be stripped.

FILES

tmp/strp*	temporary files

SEE ALSO

ar(1), **as**(1), **ld**(1), **tmpnam**(3S), **a.out**(4), **ar**(4)

NOTES

The symbol table section will not be removed if it is contained within a segment, or the file is either a relocatable or dynamic shared object.

The line number and debugging sections will not be removed if they are contained within a segment, or their associated relocation section is contained within a segment.

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**–a**] [**–g**] [*modes*]

AVAILABILITY

SUNWcsu

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (ˆ), then the value of that option is the corresponding control character (for example, ‘‘**ˆh**’’ is CTRL-H; in this case, recall that CTRL-H is the same as the ‘‘back-space’’ key.) The sequence ‘‘ˆ‘’’ means that an option has a null value.

For detailed information about the modes listed from Control Modes through Local Modes, below, see **termio**(7). For detailed information about the modes listed under Hardware Flow Control Modes and Clock Modes, below, see **termiox**(7).

Options described in the Combination Modes section are implemented using options in the earlier sections. Note: Many combinations of options make no sense, but no sanity checking is performed. Hardware flow control and clock modes options may not be supported by all hardware interfaces.

OPTIONS

–a	Report all of the option settings.
–g	Report current settings in a form that can be used as an argument to another **stty** command.

Control Modes

parenb (**–parenb**)	Enable (disable) parity generation and detection.
parext (**–parext)**	Enable (disable) extended parity generation and detection for mark and space parity.
parodd (**–parodd**)	Select odd (even) parity, or mark (space) parity if **parext** is enabled.
cs5 cs6 cs7 cs8	Select character size (see **termio**(7)).
0	Hang up line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
ispeed 0 110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal input baud rate to the number given, if possible. (Not all hardware supports split baud rates.) If the input baud rate is set to zero, the input baud rate will be specified by the value of the output baud rate.

ospeed 0 110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal output baud rate to the number given, if possible. (Not all hardware supports split baud rates.) If the output baud rate is set to zero, the line will be hung up immediately.
hupcl (–**hupcl**)	Hang up (do not hang up) connection on last close.
hup (–**hup**)	Same as **hupcl** (–**hupcl**).
cstopb (–**cstopb**)	Use two (one) stop bits per character.
cread (–**cread**)	Enable (disable) the receiver.
crtscts (**-crtscts**)	Enable hardware flow control. Raise the RTS (Request to Send) modem control line. Suspends output until the CTS (Clear to Send) line is raised.
clocal (–**clocal**)	Assume a line without (with) modem control.
loblk (–**loblk**)	Block (do not block) output from a non-current layer.
defeucw	Set the widths of multibyte Extended Unix Code (EUC) characters in struct eucioc to default values for the current locale specified by **LC_CTYPE**; width is expressed in terms of bytes per character, and screen or display columns per character (see **getwidth**(3I) and **ldterm**(7)).

Input Modes

ignbrk (–**ignbrk**)	Ignore (do not ignore) break on input.
brkint (–**brkint**)	Signal (do not signal) INTR on break.
ignpar (–**ignpar**)	Ignore (do not ignore) parity errors.
parmrk (–**parmrk**)	Mark (do not mark) parity errors (see **termio**(7)).
inpck (–**inpck**)	Enable (disable) input parity checking.
istrip (–**istrip**)	Strip (do not strip) input characters to seven bits.
inlcr (–**inlcr**)	Map (do not map) NL to CR on input.
igncr (–**igncr**)	Ignore (do not ignore) CR on input.
icrnl (–**icrnl**)	Map (do not map) CR to NL on input.
iuclc (–**iuclc**)	Map (do not map) upper-case alphabetics to lower case on input.
ixon (–**ixon**)	Enable (disable) START / STOP output control. Output is stopped by sending STOP control character and started by sending the START control character.
ixany (–**ixany**)	Allow any character (only DC1) to restart output.
ixoff (–**ixoff**)	Request that the system send (not send) START / STOP characters when the input queue is nearly empty / full.
imaxbel (–**imaxbel)**	Echo (do not echo) BEL when the input line is too long.

Output Modes

opost (–**opost**)	Post-process output (do not post-process output; ignore all other output modes).
olcuc (–**olcuc**)	Map (do not map) lower-case alphabetics to upper case on output.
onlcr (–**onlcr**)	Map (do not map) NL to CR-NL on output.
ocrnl (–**ocrnl**)	Map (do not map) CR to NL on output.
onocr (–**onocr**)	Do not (do) output CRs at column zero.
onlret (–**onlret**)	On the terminal NL performs (does not perform) the CR function.
ofill (–**ofill**)	Use fill characters (use timing) for delays.
ofdel (–**ofdel**)	Fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	Select style of delay for carriage returns (see **termio**(7)).
nl0 nl1	Select style of delay for line-feeds (see **termio**(7)).
tab0 tab1 tab2 tab3	Select style of delay for horizontal tabs (see **termio**(7)).
bs0 bs1	Select style of delay for backspaces (see **termio**(7)).
ff0 ff1	Select style of delay for form-feeds (see **termio**(7)).
vt0 vt1	Select style of delay for vertical tabs (see **termio**(7)).

Local Modes

isig (–**isig**)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (–**icanon**)	Enable (disable) canonical input (ERASE and KILL processing). Does not set MIN or TIME.
xcase (–**xcase**)	Canonical (unprocessed) upper / lower-case presentation.
echo (–**echo**)	Echo back (do not echo back) every character typed.
echoe (–**echoe**)	Echo (do not echo) ERASE character as a backspace-space-backspace string. Note: This mode will erase the ERASEed character on many CRT terminals; however, it does not keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (–**echok**)	Echo (do not echo) NL after KILL character.
lfkc (–**lfkc**)	The same as **echok** (–**echok**); obsolete.
echonl (–**echonl**)	Echo (do not echo) NL.
noflsh (–**noflsh**)	Disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (–**stwrap**)	Disable (enable) truncation of lines longer than 79 characters on a synchronous line.
tostop (–**tostop**)	Send (do not send) SIGTTOU when background processes write to the terminal.
echoctl (–echoctl)	Echo (do not echo) control characters as ^*char*, delete as ^?
echoprt (–echoprt)	Echo (do not echo) erase character as character is ‘‘erased’’.
echoke (–echoke)	BS-SP-BS erase (do not BS-SP-BS erase) entire line on line kill.

flusho (–**flusho**)	Output is (is not) being flushed.
pendin (–**pendin**)	Retype (do not retype) pending input at next read or input character.
iexten (–**iexten**)	Enable (disable) extended (implementation-defined) functions for input data.
stflush (–**stflush**)	Enable (disable) flush on a synchronous line after every **write**(2).
stappl (–**stappl**)	Use application mode (use line mode) on a synchronous line.

Hardware Flow Control Modes

rtsxoff (–**rtsxoff**)	Enable (disable) RTS hardware flow control on input.
ctsxon (–**ctsxon**)	Enable (disable) CTS hardware flow control on output.
dtrxoff (–**dtrxoff**)	Enable (disable) DTR hardware flow control on input.
cdxon (–**cdxon**)	Enable (disable) CD hardware flow control on output.
isxoff (–**isxoff**)	Enable (disable) isochronous hardware flow control on input.

Clock Modes

xcibrg	Get transmit clock from internal baud rate generator.
xctset	Get the transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
xcrset	Get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
rcibrg	Get receive clock from internal baud rate generator.
rctset	Get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
rcrset	Get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
tsetcoff	Transmitter signal element timing clock not provided.
tsetcrbrg	Output receive baud rate generator on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
tsetctbrg	Output transmit baud rate generator on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
tsetctset	Output tranmitter signal element timing (DCE source) on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
tsetcrset	Output receiver signal element timing (DCE source) on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
rsetcoff	Receiver signal element timing clock not provided.
rsetcrbrg	Output receive baud rate generator on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.

rsetctbrg	Output transmit baud rate generator on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.
rsetctset	Output transmitter signal element timing (DCE source) on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.
rsetcrset	Output receiver signal element timing (DCE source) on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.

Control Assignments

control-character c	Set *control-character* to *c*, where *control-character* is **ctab**, **discard**, **dsusp**, **eof**, **eol**, **eol2**, **erase**, **intr**, **kill**, **lnext**, **quit**, **reprint**, **start**, **stop**, **susp**, **swtch**, or **werase**. (**ctab** is used with –**stappl**, see **termio**(7)). If *c* is preceded by a caret (^) indicating an escape from the shell, then the value used is the corresponding control character (for example, ‘‘**^d**’’ is a CTRL-D). ‘‘**^?**’’ is interpreted as DEL and ‘‘**^–**’’ is interpreted as undefined.
min, time *number*	Set the value of **min** or **time** to *number.* MIN and TIME are used in Non-Canonical mode input processing (–**icanon**).
line *i*	Set line discipline to *i* (0 < *i* < 127).

Combination Modes

evenp or **parity**	Enable **parenb** and **cs7**.
oddp	Enable **parenb**, **cs7**, and **parodd**.
spacep	Enable **parenb**, **cs7**, and **parext**.
markp	Enable **parenb**, **cs7**, **parodd**, and **parext**.
–**parity**, or –**evenp**	Disable **parenb**, and set **cs8**.
–**oddp**	Disable **parenb** and **parodd**, and set **cs8**.
–**spacep**	Disable **parenb** and **parext**, and set **cs8**.
–**markp**	Disable **parenb**, **parodd**, and **parext**, and set **cs8**.
raw (–**raw** or **cooked**)	Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (–**nl**)	Unset (set) **icrnl**, **onlcr**. In addition –**nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (–**lcase**)	Set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (–**LCASE**)	Same as **lcase** (–**lcase**).
tabs (–**tabs** or **tab3**)	Preserve (expand to spaces) tabs when printing.
ek	Reset ERASE and KILL characters back to normal **#** and **@**.
sane	Resets all modes to some reasonable values.
term	Set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

async	Set normal asynchronous communications where clock settings are **xcibrg**, **rcibrg**, **tsetcoff** and **rsetcoff**.

Window Size

rows *n*	Set window size to *n* rows.
columns *n*	Set window size to *n* columns.
cols *n*	Set window size to *n* columns. Note: **cols** is a shorthand alias for columns
ypixels *n*	Set vertical window size to *n* pixels.
xpixels *n*	Set horizontal window size to *n* pixels.

SEE ALSO

tabs(1), **ioctl**(2), **getwidth**(3I), **ldterm**(7), **termio**(7), **termiox**(7)

NAME

stty – set the options for a terminal

SYNOPSIS

/usr/ucb/stty [–**a**] [–**g**] [–**h**] [*modes*]

AVAILABILITY

SUNWscpu

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard output; without arguments, it reports the settings of certain options.

OPTIONS

In this report, if a character is preceded by a caret (ˆ), then the value of that option is the corresponding CTRL character (for example, ‘‘**ˆh**’’ is **CTRL–h;** in this case, recall that **CTRL–h** is the same as the ‘‘back-space’’ key.) The sequence ‘‘**ˆ´**’’ means that an option has a null value.

–a	Report all of the option settings.
–g	Report current settings in a form that can be used as an argument to another *stty* command.
–h	Report all the option settings with the control characters in an easy to read column format.

Options in the last group are implemented using options in the previous groups. Note: Many combinations of options make no sense, but no sanity checking is performed. Hardware flow control and clock modes options may not be supported by all hardware interfaces. The options are selected from the following:

Special Requests

all	Reports the same option settings as **stty** without arguments, but with the control characters in column format.
everything	Everything **stty** knows about is printed. Same as –**h** option.
speed	The terminal speed alone is reported on the standard output.
size	The terminal (window) sizes are printed on the standard output, first rows and then columns. This option is only appropriate if currently running a window system. **size** and **speed** always report on the settings of **/dev/tty**, and always report the settings to the standard output.

Control Modes

parenb (–**parenb**)	Enable (disable) parity generation and detection.
parext (–**parext)**	Enable (disable) extended parity generation and detection for mark and space parity.
parodd (–**parodd**)	Select odd (even) parity, or mark (space) parity if **parext** is enabled.
cs5 cs6 cs7 cs8	Select character size (see **termio**(7)).
0	Hang up line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 exta 38400 extb	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
ispeed 0 110 300 600 1200 1800 2400 4800 9600 19200 exta 38400 extb	Set terminal input baud rate to the number given, if possible. (Not

all hardware supports split baud rates.) If the input baud rate is set to zero, the input baud rate will be specified by the value of the output baud rate.

ospeed 0 110 300 600 1200 1800 2400 4800 9600 19200 exta 38400 extb

Set terminal output baud rate to the number given, if possible. (Not all hardware supports split baud rates.) If the baud rate is set to zero, the line will be hung up immediately.

hupcl (–**hupcl**)	Hang up (do not hang up) connection on last close.
hup (–**hup**)	Same as **hupcl** (–**hupcl**).
cstopb (–**cstopb**)	Use two (one) stop bits per character.
cread (–**cread**)	Enable (disable) the receiver.
clocal (–**clocal**)	Assume a line without (with) modem control.
crtscts (**-crtscts**)	Enable hardware flow control. Raise the RTS (Request to Send) modem control line. Suspends output until the CTS (Clear to Send) line is raised.
loblk (–**loblk**)	Block (do not block) output from a non-current layer.

Input Modes

ignbrk (–**ignbrk**)	Ignore (do not ignore) break on input.
brkint (–**brkint**)	Signal (do not signal) INTR on break.
ignpar (–**ignpar**)	Ignore (do not ignore) parity errors.
parmrk (–**parmrk**)	Mark (do not mark) parity errors (see **termio**(7)).
inpck (–**inpck**)	Enable (disable) input parity checking.
istrip (–**istrip**)	Strip (do not strip) input characters to seven bits.
inlcr (–**inlcr**)	Map (do not map) NL to CR on input.
igncr (–**igncr**)	Ignore (do not ignore) CR on input.
icrnl (–**icrnl**)	Map (do not map) CR to NL on input.
iuclc (–**iuclc**)	Map (do not map) upper-case alphabetics to lower case on input.
ixon (–**ixon**)	Enable (disable) START / STOP output control. Output is stopped by sending an STOP and started by sending an START.
ixany (–**ixany**)	Allow any character (only START) to restart output.
decctlq (–**decctlq**)	Same as –**ixany**.
ixoff (–**ixoff**)	Request that the system send (not send) START / STOP characters when the input queue is nearly empty / full.
tandem (–tandem)	Same as **ixoff**.
imaxbel (–imaxbel)	Echo (do not echo) BEL when the input line is too long.
iexten (–iexten)	Enable (disable) extended (implementation-defined) functions for input data.

Output Modes

opost (–**opost**)	Post-process output (do not post-process output; ignore all other output modes).
olcuc (–**olcuc**)	Map (do not map) lower-case alphabetics to upper case on output.
onlcr (–**onlcr**)	Map (do not map) NL to CR-NL on output.
ocrnl (–**ocrnl**)	Map (do not map) CR to NL on output.
onocr (–**onocr**)	Do not (do) output CRs at column zero.
onlret (–**onlret**)	On the terminal NL performs (does not perform) the CR function.

ofill (−**ofill**)	Use fill characters (use timing) for delays.
ofdel (−**ofdel**)	Fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	Select style of delay for carriage returns (see **termio**(7)).
nl0 nl1	Select style of delay for line-feeds (see **termio**(7)).
tab0 tab1 tab2 tab3	Select style of delay for horizontal tabs (see **termio**(7)).
bs0 bs1	Select style of delay for backspaces (see **termio**(7)).
ff0 ff1	Select style of delay for form-feeds (see **termio**(7)).
vt0 vt1	Select style of delay for vertical tabs (see **termio**(7)).

Local Modes

isig (−**isig**)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (−**icanon**)	Enable (disable) canonical input (ERASE and KILL processing). Does not set MIN or TIME.
cbreak (−**cbreak**)	Equivalent to **-icanon min 1 time 0**.
xcase (−**xcase**)	Canonical (unprocessed) upper / lower-case presentation.
echo (−**echo**)	Echo back (do not echo back) every character typed.
echoe (−**echoe**)	Echo (do not echo) ERASE character as a backspace-space-backspace string. Note: This mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
crterase (−**crterase**)	Same as **echoe**.
echok (−**echok**)	Echo (do not echo) NL after KILL character.
lfkc (−**lfkc**)	The same as **echok** (−**echok**); obsolete.
echonl (−**echonl**)	Echo (do not echo) NL.
noflsh (−**noflsh**)	Disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (−**stwrap**)	Disable (enable) truncation of lines longer than 79 characters on a synchronous line. (Does not apply to the 3B2.)
tostop (−**tostop**)	Send (do not send) SIGTTOU for background processes.
echoctl (−**echoctl)**	Echo (do not echo) control characters as ˆ*char*, delete as ˆ?
ctlecho (−**ctlecho)**	Same as **echoctl**.
echoprt (−**echoprt)**	Echo (do not echo) erase character as character is ‘‘erased’’.
prterase (−**prterase**)	Same as **echoprt**.
echoke (−**echoke)**	BS-SP-BS erase (do not BS-SP-BS erase) entire line on line kill.
crtkill (−**crtkill**)	Same as **echoke**.
flusho (−**flusho)**	Output is (is not) being flushed.
pendin (−**pendin)**	Retype (do not retype) pending input at next read or input character.
stflush (−**stflush**)	Enable (disable) flush on a synchronous line after every **write**(2). (Does not apply to the 3B2.)
stappl (−**stappl**)	Use application mode (use line mode) on a synchronous line. (Does not apply to the 3B2.)

Hardware Flow Control Modes

rtsxoff (−**rtsxoff**)	Enable (disable) RTS hardware flow control on input.
ctsxon (−**ctsxon**)	Enable (disable) CTS hardware flow control on output.
dterxoff (−**dterxoff**)	Enable (disable) DTER hardware flow control on input.
rlsdxon (−**rlsdxon**)	Enable (disable) RLSD hardware flow control on output.
isxoff (−**isxoff**)	Enable (disable) isochronous hardware flow control on input.

Clock Modes

xcibrg	Get transmit clock from internal baud rate generator.
xctset	Get the transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
xcrset	Get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
rcibrg	Get receive clock from internal baud rate generator.
rctset	Get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
rcrset	Get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
tsetcoff	Transmitter signal element timing clock not provided.
tsetcrc	Output receive clock on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24, clock source.
tsetcxc	Output transmit clock on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24, clock source.
rsetcoff	Receiver signal element timing clock not provided.
rsetcrc	Output receive clock on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin, clock source.
rsetcxc	Output transmit clock on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin, clock source.

Control Assignments

control-character c	Set *control-character* to *c*, where *control-character* is **intr**, **quit**, **erase**, **kill**, **eof**, **eol**, **eol2**, **swtch**, **start**, **stop**, **susp**, **dsusp**, **rprnt**, **flush**, **werase**, **lnext min**, **ctab**, **time**, or **brk**) (**ctab** is used with −**stappl**; **min** and **time** are used with −**icanon**; see **termio**(7)). If *c* is preceded by an (escaped from the shell) caret (**^**), then the value used is the corresponding CTRL character (for example, ‘‘**^d**’’ is a **CTRL-d**); ‘‘**^?**’’ is interpreted as DEL and ‘‘**^−**’’ is interpreted as undefined.
line *i*	Set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity**	Enable **parenb** and **cs7**.
−**evenp**, or −**parity**	Disable **parenb**, and set **cs8**.
even (−**even**)	Same as **evenp** (−**evenp**).
oddp	Enable **parenb**, **cs7**, and **parodd**.

–oddp	Disable **parenb** and **parodd**, and set **cs8**.
odd (–**odd**)	Same as **oddp** (–**oddp**).
spacep	Enable **parenb**, **cs7**, and **parext**.
–spacep	Disable **parenb** and **parext**, and set **cs8**.
markp	Enable **parenb**, **cs7**, **parodd**, and **parext**.
–markp	Disable **parenb**, **parodd**, and **parext**, and set **cs8**.
raw (–**raw** or **cooked**)	Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (–**nl**)	Unset (set) **icrnl**, **onlcr**. In addition –**nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (–**lcase**)	Set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (–**LCASE**)	Same as **lcase** (–**lcase**).
tabs (–**tabs** or **tab3**)	Preserve (expand to spaces) tabs when printing.
ek	Reset ERASE and KILL characters back to normal **#** and **@**.
sane	Resets all modes to some reasonable values.
term	Set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.
async	Set normal asynchronous communications where clock settings are **xcibrg**, **rcibrg**, **tsetcoff** and **rsetcoff**.
litout (–litout)	Disable (enable) **parenb**, **istrip**, and **opost**, and set **cs8** (**cs7**).
pass8 (–pass8)	Disable (enable) **parenb** and **istrip**, and set **cs8** (**cs7**).
crt	Set options for a CRT (**echoe**, **echoctl**, and, if >= 1200 baud, **echoke**.)
dec	Set all modes suitable for Digital Equipment Corp. operating systems users (**ERASE**, **KILL**, and **INTR** characters to **ˆ?**, **ˆU**, and **ˆC**, **decctlq**, and **crt**.)

Window Size

rows*n*	Set window size to *n rows.*
columns*n*	Set window size to *n columns.*
cols*n*	An alias for **columns** *n*.
ypixels*n*	Set vertical window size to *n pixels.*
xpixels*n*	Set horizontal window size to *n pixels.*

SEE ALSO

tabs(1), **ioctl**(2), **termio**(7), **termiox**(7)

NAME

sum – print checksum and block count for a file

SYNOPSIS

sum [**–r**] *filename*

AVAILABILITY

SUNWesu

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of 512 byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

OPTIONS

–r Use an alternate (machine-dependent) algorithm in computing the checksum.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE**, **LC_MESSAGES**, **LC_TIME**, **LC_COLLATE**, **LC_NUMERIC**, and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **sum** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **sum** behaves.

LC_CTYPE

Determines how **sum** handles characters. When **LC_CTYPE** is set to a valid value, **sum** can display and handle text and filenames containing valid characters for that locale. **sum** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **sum** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

wc(1), **environ**(5)

DIAGNOSTICS

‘‘Read error’’ is indistinguishable from end of file on most devices; check the block count.

NAME

sum – calculate a checksum for a file

SYNOPSIS

/usr/ucb/sum *filename*

AVAILABILITY

SUNWscpu

DESCRIPTION

sum calculates and displays a 16-bit checksum for the named file, and also displays the size of the file in kilobytes. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The checksum is calculated by an algorithm which may yield different results on machines with 16-bit **int**s and machines with 32-bit **int**s, so it cannot always be used to validate that a file has been transferred between machines with different-sized **int**s.

SEE ALSO

sum(1), **wc**(1)

DIAGNOSTICS

Read error is indistinguishable from EOF on most devices; check the block count.

NOTES

Obsolete.

NAME

suspend – shell built-in function to halt the current shell

SYNOPSIS

sh

suspend

csh

suspend

ksh

suspend

DESCRIPTION

sh

Stops the execution of the current shell (but not if it is the login shell).

csh

Stop the shell in its tracks, much as if it had been sent a stop signal with **ˆZ**. This is most often used to stop shells started by **su**.

ksh

Stops the execution of the current shell (but not if it is the login shell).

SEE ALSO

csh(1), **ksh**(1), **kill**(1), **sh**(1), **su**(1)

NAME

symorder – rearrange a list of symbols

SYNOPSIS

symorder [**–s**] *objectfile symbolfile*

DESCRIPTION

This program was specifically designed to cut down on the overhead of getting symbols from **/kernel/unix**.

objectfile is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

symbolfile is a file containing symbols to be found in *objectfile*, one symbol per line.

OPTIONS

–s Work silently, that is, display nothing except error messages. This is useful for checking the error status.

FILES

/kernel/unix

SEE ALSO

nlist(3E)

NAME

sysV-make – maintain, update, and regenerate groups of programs

SYNOPSIS

/usr/ccs/lib/svr4.make [–f *makefile***] [–eiknpqrst] [** *names* **]**

DESCRIPTION

This is the ‘‘vanilla’’ System V version of **make**. If the environment variable **USE_SVR4_MAKE** is set, then the command **make** will invoke this version of **make**. (See also the **ENVIRONMENT** section.)

make allows the programmer to maintain, update, and regenerate groups of computer programs. **make** executes commands in *makefile* to update one or more target *names* (*names* are typically programs). If the **–f** option is not present, then **makefile**, **Makefile**, and the Source Code Control System (SCCS) files **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is –, the standard input is taken. More than one **–f** *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be outdated.

The following list of four directives can be included in *makefile* to extend the options provided by **make**. They are used in *makefile* as if they were targets:

.DEFAULT:	If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.
.IGNORE:	Same effect as the **–i** option.
.PRECIOUS:	Dependents of the **.PRECIOUS** entry will not be removed when quit or interrupt are hit.
.SILENT:	Same effect as the **–s** option.

The options for **make** are listed below:

–e	Environment variables override assignments within makefiles.
–f *makefile*	Description filename (*makefile* is assumed to be the name of a description file).
–i	Ignore error codes returned by invoked commands.
–k	Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.
–n	No execute mode. Print commands, but do not execute them. Even command lines beginning with an **@** are printed.
–p	Print out the complete set of macro definitions and target descriptions.
–q	Question. **make** returns a zero or non-zero status code depending on whether or not the target file has been updated.
–r	Do not use the built-in rules.
–s	Silent mode. Do not print command lines before executing.

–t Touch the target files (causing them to be updated) rather than issue the usual commands.

Creating the makefile

The makefile invoked with the **–f** option is a carefully structured file of explicit instructions for updating and regenerating programs, and contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a **:**, then a (possibly null) list of prerequisite files or dependencies. Text following a **;** and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or **#** begins a new dependency or macro definition. Shell commands may be continued across lines with a backslash-new-line (\ new-line) sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (**#**) and new-line surround comments including contained \ new-line sequences.

The following makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o –o pgm
a.o: incl.h a.c
        cc –c a.c
b.o: incl.h b.c
        cc –c b.c
```

Command lines are executed one at a time, each by its own shell. The **SHELL** environment variable can be used to specify which shell **make** should use to execute commands. The default is **/usr/bin/sh**. The first one or two characters in a command can be the following: **@**, –, **@**–, or –**@**. If **@** is present, printing of the command is suppressed. If – is present, **make** ignores an error. A line is printed when it is executed unless the **–s** option is present, or the entry **.SILENT:** is included in *makefile*, or unless the initial character sequence contains a **@**. The **–n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see the discussion of the **MAKEFLAGS** macro in the ‘‘make Environment’’ section below). The **–t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate **make**. If the **–i** option is present, if the entry **.IGNORE:** is included in *makefile*, or if the initial character sequence of the command contains –, the error is ignored. If the **–k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the directive **.PRECIOUS**.

make Environment

The environment is read by **make**. All variables are assumed to be macro definitions and are processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The −**e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by **make** as containing any legal input option (except −**f** and −**p**) defined for the command line. Further, upon invocation, **make** ‘‘invents’’ the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This feature proves very useful for ‘‘super-makes’’. In fact, as noted above, when the −**n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make** −**n** recursively on a whole software system to see what would have been executed. This result is possible because the −**n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This usage is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string **include** appears as the first seven letters of a line in a *makefile*, and is followed by a blank or a tab, the rest of the line is assumed to be a filename and will be read by the current invocation, after substituting for any macros.

Macros

Entries of the form *string1* **=** *string2* are macro definitions. *string2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of **$**(*string1*[**:***subst1***=**[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single-character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown in the ‘‘Libraries’’ section below.

Internal Macros

There are five internally maintained macros that are useful for writing rules for building targets.

$∗** The macro **$∗ stands for the filename part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module that is outdated with respect to the target (the ‘‘manufactured’’ dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc −c −O $*.c
```

or:

```
.c.o:
        cc −c −O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are outdated with respect to the target, and essentially those modules that must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to ‘‘directory part’’ for **D** and ‘‘file part’’ for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, **make** has inference rules that allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c     .c~     .f     .f~    .s     .s~    .sh    .sh~   .C     .C~
.c.a   .c.o    .c~.a  .c~.c  .c~.o  .f.a   .f.o   .f~.a  .f~.f  .f~.o
.h~.h  .l.c    .l.o   .l~.c  .l~.l  .l~.o  .s.a   .s.o   .s~.a  .s~.o
.s~.s  .sh~.sh .y.c   .y.o   .y~.c  .y~.o  .y~.y  .C.a   .C.o   .C~.a
.C~.C  .C~.o   .L.C   .L.o   .L~.C  .L~.L  .L~.o  .Y.C   .Y.o   .Y~.C
.Y~.o  .Y~.Y
```

The internal rules for **make** are contained in the source file **rules.c** for the **make** program. These rules can be locally modified. To print out the rules compiled into the **make** on any machine in a form suitable for recompilation, the following command is used:

```
make −pf − 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file (see **sccsfile**(4)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with the **make** suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (for example, **.c:**) is the definition of how to build *x* from *x*.**c**. In effect, the other suffix is null. This feature is useful for building targets from only one source file, for example, shell procedures and simple C programs.

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant: the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .c˜ .y .y˜ .l .l˜ .s .s˜ .sh .sh˜ .h .h˜ .f .f˜ .C .C˜ .Y .Y˜ .L .L˜

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
        cc a.o b.o –o pgm
a.o b.o: incl.h
```

This abbreviation is possible because **make** has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to **cc**(1B). Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus, **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library that contains **file.o**. (This example assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form **.***XX***.a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate by-product of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) –c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm –f $*.o
```

In fact, the **.c.a** rule listed above is built into **make** and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) –c $(CFLAGS) $(?:.o=.c)
        $(AR) $(ARFLAGS) lib $?
        rm $?
        @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object filenames (inside **lib**) whose C source files are outdated. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c˜**; however, this transformation may become possible in the future.) Also note the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

ENVIRONMENT

USE_SVR4_MAKE

If this environment variable is set, then the **make** command will invoke this System V version of **make**. If this variable is not set, then the default version of **make**(1S) is invoked.

USE_SVR4_MAKE can be set as follows (Bourne shell):

```
$ USE_SVR4_MAKE=“”; export USE_SVR4_MAKE
```

or (C shell):

```
% setenv USE_SVR4_MAKE
```

FILES

[Mm]akefile and **s.[Mm]akefile**
/usr/bin/sh

SEE ALSO

cc(1B), **cd**(1), **make**(1S), **sh**(1), **printf**(3S), **sccsfile**(4)

Programming Utilities Guide

NOTES

Some commands return non-zero status inappropriately; use **–i** or the – command line prefix to overcome the difficulty.

Filenames with the characters **= : @** will not work. Commands that are directly executed by the shell, notably **cd**(1), are ineffectual across new-lines in **make**. The syntax **lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**.

NAME

tabs – set tabs on a terminal

SYNOPSIS

tabs [*tabspec*] [**–T***type*] [**+m***n*]

AVAILABILITY

SUNWcsu

DESCRIPTION

tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely settable hardware tabs.

OPTIONS

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (*–code*), repetitive (*–n*), arbitrary (*n1,n2,...*), and file (*––file*). If no *tabspec* is given, the default value is –**8**, that is, UNIX system ''standard'' tabs. The lowest column number is 1. Note: For **tabs**, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, for example, the DASI 300, DASI 300s, and DASI 450.

–code Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

–a	1,10,16,36,72 Assembler, IBM S / 370, first format
–a2	1,10,16,40,72 Assembler, IBM S / 370, second format
–c	1,8,12,16,20,55 COBOL, normal format
–c2	1,6,10,14,49 COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see **fspec**(4)): **<:t–c2 m6 s66 d:>**
–c3	1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67 COBOL compact format (columns 1-6 omitted), with more tabs than **–c2.** This is the recommended format for COBOL. The appropriate format specification is (see **fspec**(4)): **<:t–c3 m6 s66 d:>**
–f	1,7,11,15,19,23 FORTRAN
–p	1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61 PL / I
–s	1,10,55 SNOBOL
–u	1,12,20,44 UNIVAC 1100 Assembler

–n	A *repetitive* specification requests tabs at columns 1+*n*, 1+2∗*n*, etc. Of particular importance is the value **8**: this represents the UNIX system ‘‘standard’’ tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0**, implying no tabs at all.
n1,n2,...	The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1**,**10**,**20**,**30**, and **1**,**10**,**+10**,**+10** are considered identical.
––file	If the name of a *file* is given, **tabs** reads the first line of the file, searching for a format specification (see **fspec**(4)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as **–8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the **pr** command:

```
example% tabs –– file; pr file
```

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

–T*type*	**tabs** usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in **term**(5). If no **–T** flag is supplied, **tabs** uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* (see **environ**(5)), **tabs** tries a sequence that will work for many terminals.
+m*n*	The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is **10**. For a TermiNet, the first value in the tab list should be **1**, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

The command:

```
example% tabs –a
```

is an example using *–code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

The next command:

```
example% tabs –8
```

is an example of using *–n* (*repetitive* specification), where *n* is **8**, causes tabs to be set every eighth position:
1+(1∗8), 1+(2∗8), . . . which evaluate to columns 9, 17, . . .

The command:

```
example% tabs 1,8,36
```

is an example of using *n1*,*n2*,. . . (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

The last command:

```
example% tabs —$HOME/fspec.list/att4425
```

is an example of using −−*file* (*file* specification) to indicate that tabs should be set according to the first line of **$HOME/fspec.list/att4425** (see **fspec**(4)).

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tabs** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tabs** behaves.

LC_CTYPE

Determines how **tabs** handles characters. When **LC_CTYPE** is set to a valid value, **tabs** can display and handle text and filenames containing valid characters for that locale. **tabs** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tabs** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

newform(1), **pr**(1), **tput**(1), **fspec**(4), **terminfo**(4), **environ**(5), **term**(5)

DIAGNOSTICS

illegal tabs	This occurs when arbitrary tabs are ordered incorrectly.
illegal increment	This occurs when a zero or missing increment is found in an arbitrary specification.
unknown tab code	This occurs when a *canned* code cannot be found.
can't open	The —*file* option was used, and file cannot be opened.
file indirection	The —*file* option was used and the specification in that file points to yet another file. Indirection of this form is not permitted.

NOTES

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

The *tabspec* used with the **tabs** command is different from the one used with the **newform** command. For example, **tabs –8** sets every eighth position; whereas **newform –i–8** indicates that tabs are set every eighth position.

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [± *number* [**lbcr**]] [*filename*]
tail [–**lbcr**] [*filename*]
tail [± *number* [**lbcf**]] [*filename*]
tail [–**lbcf**] [*filename*]

AVAILABILITY

SUNWcsu

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance **+***number* from the beginning, or –*number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

The **r** and **f** options are mutually exclusive. If both are specified on the command line, the **f** option will be ignored.

OPTIONS

–**l**	Units of lines.
–**b**	Units of blocks.
–**c**	Units of characters.
–**f**	With the –**f** (follow) option, if the input-file is not a pipe, the program will not terminate after the line of the input-file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input-file. Thus it may be used to monitor the growth of a file that is being written by some other process.
–**r**	The **r** option copies lines from the specified starting point in the file in reverse order. The default for **r** is to print the entire file in reverse order.

EXAMPLES

For example, the command:

```
example% tail –f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed. As another example, the command:

```
example% tail –15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tail** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tail** behaves.

LC_CTYPE
Determines how **tail** handles characters. When **LC_CTYPE** is set to a valid value, **tail** can display and handle text and filenames containing valid characters for that locale. **tail** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tail** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

cat(1), **head**(1), **more**(1), **pg**(1), **tail**(1), **dd**(1M), **environ**(5)

NOTES

Piped tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

talk – talk to another user

SYNOPSIS

talk *username* [*ttyname*]

AVAILABILITY

SUNWcsu

DESCRIPTION

talk is a visual communication program that copies lines from your terminal to that of a user on the same or on another host. *username* is that user's login name.

The program is architecture dependent; it works only between machines of the same architecture.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, **talk** sends the message:

Message from TalkDaemon@ *her_machine* **at** *time* . . .
talk: connection requested by *your_name***@***your_machine*
talk: respond with: talk *your_name***@***your_machine*

to the user you wish to talk to. At this point, the recipient of the message should reply by typing:

talk *your_name***@***your_machine*

It does not matter from which machine the recipient replies, as long as the login name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing Ctrl-L redraws the screen, while your erase, kill, and word kill characters will work in **talk** as normal. To exit, just type your interrupt character; **talk** then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the **mesg**(1) command. At the outset talking is allowed. Certain commands, such as **pr**(1), disallow messages in order to prevent messy output.

FILES

/etc/hosts	to find the recipient's machine
/var/adm/utmp	to find the recipient's tty

SEE ALSO

mail(1), **mesg**(1), **pr**(1), **who**(1), **write**(1)

NAME

tar – create tape archives, and add or extract files

SYNOPSIS

/usr/sbin/tar c [**bBefFhilvwX** [**0-7**]] [*device*] [*block*] [*exclude-filename* ...]
[**–I** *include-filename*] *filename* ... [**–C** *directory filename*]

/usr/sbin/tar r [**bBefFhilvw** [**0-7**]] [*device*] [*block*] [**–I** *include-filename*]
filename ... [**–C** *directory filename*]

/usr/sbin/tar t [**BefFhilvX** [**0-7**]] [*device*] [*exclude-filename* ...]
[**–I** *include-filename*] [*filename* ...]

/usr/sbin/tar u [**bBefFhilvw** [**0-7**]] [*device*] [*block*] *filename* ...

/usr/sbin/tar x [**BefFhilmopvwX** [**0-7**]] [*device*] [*exclude-filename* ...] [*filename* ...]

DESCRIPTION

tar archives and extracts files to and from a single file called a *tarfile* . A tarfile is usually a magnetic tape, but it can be any file. **tar**’s actions are controlled by the *key* argument. The *key* is a string of characters containing exactly one function letter (**c**, **r**, **t** , **u**, or **x**) and one or more function modifiers, depending on the function letter used. Other arguments to the command are *filename*s (or directory names) specifying which files are to be archived or extracted. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

Function Letters

The function portion of the key is specified by one of the following letters:

c Create. Writing begins at the beginning of the tarfile, instead of at the end. This key implies the **r** key.

r Replace. The named *filename*s are written on the end of the tape. The **c** and **u** functions imply this function. See **NOTES** for more information.

t Table of Contents. The names of the specified files are listed each time they occur on the *tarfile*. If no *filename*s arguments are given, all the names on the *tarfile* are listed. With the **v** function modifier, additional information for the specified files is displayed. The listing is similar to the format produced by the **ls -l** command.

u Update. The named *filename*s are added to the tarfile if they are not already there, or have been modified since last written on that tarfile. This key implies the **r** key. See **NOTES** for more information.

x Extract, or restore. The named *filename*s are extracted from the tarfile and written to the current directory. If a named file matches a directory whose contents had been written onto the tarfile, this directory is (recursively) extracted. Use the file or directory’s relative path when appropriate, or **tar** will not find a match. The owner, modification time, and mode are restored (if possible); otherwise, to restore owner, you must be the superuser. If no *filename*s argument is given, the entire content of the tarfile is extracted. Note: If several files with the same name are on the tarfile, the last one overwrites all earlier ones. See **NOTES** for more information.

Function Modifiers

The characters below may be used in addition to the letter that selects the desired function. Use them in the order shown in the **SYNOPSIS.**

b Blocking Factor. This causes **tar** to use the *block* argument as the blocking factor for tape records. The default is **1**, the maximum is **20**. This function should not be supplied when operating on regular archives or block special devices. It is mandatory however, when reading archives on raw magnetic tape archives (see **f** below). The block size is determined automatically when reading tapes created on block special devices (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to **tar** when it is run. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

B Block. Force **tar** to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that **tar** can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming. When reading from standard input, '**-**', this option is automatically selected to make sure that **tar** can recover from short reads.

e Error. If any unexpected errors occur **tar** will exit immediately with a positive exit status.

f File. This causes **tar** to use the *device* argument as the name of the tarfile. If **f** is given, **/etc/default/tar** is not searched. If **f** is omitted, **tar** will use the device indicated by the **TAPE** environment variable, if set; otherwise, it will use the default values defined in **/etc/default/tar**. If the name of the tarfile is '–', **tar** writes to the standard output or reads from the standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a pipeline. **tar** can also be used to move hierarchies with the command:

```
example% cd fromdir; tar cf – . | (cd todir; tar xfBp –)
```

F With one **F** argument, **tar** will exclude all directories named SCCS from the tarfile. With two arguments, **FF , tar** will exclude all directories named SCCS, all files with **.o** as their suffix, and all files named **errs**, **core**, and **a.out**.

h Follow symbolic links as if they were normal files or directories. Normally, **tar** does not follow symbolic links.

i Ignore. With this option **tar** will ignore directory checksum errors.

l Link. This tells **tar** to complain if it cannot resolve all of the links to the files being archived. If **l** is not specified, no error messages are printed.

m Modify. This tells **tar** to not extract the modification times from the tarfile. The modification time of the file will be the time of extraction. This option is only valid with the **x** key.

o Ownership. This causes extracted files to take on the user and group identifier of the user running the program, rather than those on tape. This happens by default for users other than root. If the 'o' option is not set and the user is root, the extracted files will take on the group and user identifiers of the files on tape (see **chown**(1) for more information). The 'o' option is only valid with the **x** key.

p Restore the named files to their original modes, ignoring the present **umask**(2). SETUID and sticky information are also extracted if your are the super-user. This option is only useful with the **x** key letter.

v Verbose. Normally, **tar** does its work silently. This option causes **tar** to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w What. This option causes **tar** to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means no. This is not valid with the **t** key.

X Exclude. Use the *exclude-filename* argument as a file containing a list of named files (or directories) to be excluded from the tarfile when using the key letters **c**, **x**, or **t**. Multiple **X** arguments may be used, with one *exclude-filename* per argument. See **NOTES** for more information.

[0-7] Select an alternative drive on which the tape is mounted. The default entries are specified in **/etc/default/tar**.

If a file name is preceded by –**I** then the filename is opened. A list filenames, one per line, is treated as if each appeared separately on the command line. Be careful of trailing white space in both include and exclude file lists.

In the case where excluded files (see **X** option) also exist, excluded files take precedence over all included files. So, if a file is specified in both the include and exclude files (or on the command line), it will be excluded.

If a file name is preceded by –**C** in a **c** (create) or **r** (replace) operation, **tar** will perform a **chdir** (see **csh**(1)) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names.

Note: the –**C** option only applies to *one* following directory name and *one* following file name.

If no digit or 'f' is given, the entry in **/etc/default/tar** with digit "0" will be the default.

EXAMPLES

To archive files from **/usr/include** and from **/etc**, onto default tape drive **0** one might use:

```
example% tar c –C /usr  include –C /etc .
```

If you get a table of contents from the resulting *tarfile*, you might see something like:

```
include/
include/a.out.h
and all the other files in /usr/include . . .
/chown
and all the other files in /etc
```

To extract all files under **include**:

```
example% tar xv include
x include/, 0 bytes, 0 tape blocks
and all files under include. . .
```

Here is a simple example using **tar** to create an archive of your home directory on a tape mounted on drive **/dev/rmt/0**:

```
example% cd
example% tar cvf /dev/rmt/0 .
messages from tar
```

The **c** option means create the archive; the **v** option makes **tar** tell you what it is doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt/0** in this example).

Now you can read the table of contents from the archive like this:

```
example% tar tvf /dev/rmt/0
rw-r--r--    1677/40    2123    Nov  7 18:15 1985        ./test.c
. . .
example%
```

The columns have the following meanings:

- column 1 is the access permissions to **./test.c**
- column 2 is the *user-id* / *group-id* of **./test.c**
- column 3 is the size of **./test.c** in bytes
- column 4 is the modification date of **./test.c**
- column 5 is the name of **./test.c**

You can extract files from the archive like this:

```
example% tar xvf /dev/rmt/0
messages from tar
example%
```

If there are multiple archive files on a tape, each is separated from the following one by an EOF marker. **tar** does not read the EOF mark on the tape after it finishes reading an archive file because **tar** looks for a special header to decide when it has reached the end of the archive. Now if you try to use **tar** to read the next archive file from the tape, **tar** does not know enough to skip over the EOF mark and tries to read the EOF mark as an archive instead. The result of this is an error message from **tar** to the effect:

```
tar: blocksize=0
```

This means that to read another archive from the tape, you must skip over the EOF marker before starting another **tar** command. You can accomplish this using the **mt**(1) command, as shown in the example below. Assume that you are reading from **/dev/rmt/0n**.

```
example% tar xvfp /dev/rmt/0n      read first archive from tape
messages from tar
example% mt fsf 1                  skip over the end-of-file marker
example% tar xvfp /dev/rmt/0n      read second archive from tape
messages from tar
example%
```

Finally, here is an example using **tar** to transfer files across the Ethernet. First, here is how to archive files from the local machine (**example**) to a tape on a remote system (**host**):

```
example% tar cvfb – 20 filenames | rsh host dd of=/dev/rmt/0 obs=20b
messages from tar
example%
```

In the example above, we are *creating* a *tarfile* with the **c** key letter, asking for *verbose* output from **tar** with the **v** option, specifying the name of the output *tarfile* using the **f** option (the standard output is where the *tarfile* appears, as indicated by the '–' sign), and specifying the blocksize (**20**) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the **tar** command *and* on the **dd** command.

Now, here is how to use **tar** to get files from a tape on the remote system back to the local system:

```
example% rsh –n host dd if=/dev/rmt/0 bs=20b | tar xvBfb – 20 filenames
messages from tar
example%
```

In the example above, we are *extracting* from the *tarfile* with the **x** key letter, asking for *verbose output from* **tar** with the **v** option, telling **tar** it is reading from a pipe with the **B** option, specifying the name of the input *tarfile* using the **f** option (the standard input is where the *tarfile* appears, as indicated by the '–' sign), and specifying the blocksize (**20**) with the **b** option.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tar** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tar** behaves.

LC_CTYPE

Determines how **tar** handles characters. When **LC_CTYPE** is set to a valid value, **tar** can display and handle text and filenames containing valid characters for that locale. **tar** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tar** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME
Determines how **tar** handles date and time formats. In the "C" locale, date and time handling follow the U.S. rules.

FILES

/dev/rmt/[0-7][b][n]
/dev/rmt/[0-7]l[b][n]
/dev/rmt/[0-7]m[b][n]
/dev/rmt/[0-7]h[b][n]
/dev/rmt/[0-7]u[b][n]
/dev/rmt/[0-7]c[b][n]

/etc/default/tar Settings may look like this:

```
archive0=/dev/rmt/0
archive1=/dev/rmt/0n
archive2=/dev/rmt/1
archive3=/dev/rmt/1n
archive4=/dev/rmt/0
archive5=/dev/rmt/0n
archive6=/dev/rmt/1
archive7=/dev/rmt/1n
```

/tmp/tar*

SEE ALSO

ar(1), **chown**(1), **cpio**(1), **csh**(1), **ls**(1), **mt**(1), **umask**(2), **environ**(5)

DIAGNOSTICS

Complaints about bad key characters and tape read / write errors.
Complaints if enough memory is not available to hold the link tables.

NOTES

There is no way to ask for the *n*-th occurrence of a file.

Tape errors are handled ungracefully.

The **u** option can be slow.

The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

When extracting tapes created with the **r** or **u** option, directory modification times may not be set correctly.

When using **r**,**u**,**x**, or**X**, the named files must match exactly to the corresponding files in the *tarfile*. For example, to extract **./***filename*, you must specify **./***filename*, and not *filename*. The **t** option displays how each file was archived.

The full pathname length cannot exceed 255 characters.

The file name (or leaf) length cannot exceed 100 characters.

The prefix of the pathname cannot exceed 155 characters.

tar does not copy empty directories or special files such as devices.

Filename substitution wildcards do not work for extracting files from the archive. To get around this, use a command of the form:

```
tar xvf... /dev/rmt/0 ‘tar tf... /dev/rmt/0 | grep ’pattern’‘
```

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [**–ms**] [*filename*] . . .

AVAILABILITY

SUNWdoc

DESCRIPTION

tbl is a preprocessor for formatting tables for **nroff**(1) or **troff**(1). The input *filename*s are copied to the standard output, except that lines between **.TS** and **.TE** command lines are assumed to describe tables and are reformatted.

If no arguments are given, **tbl** reads the standard input, so **tbl** may be used as a filter. When **tbl** is used with **eqn**(1) or **neqn** the **tbl** command should be first, to minimize the volume of data passed through pipes.

OPTIONS

–ms Copy the **–ms** macro package to the front of the output file.

EXAMPLES

As an example, letting **\t** represent a TAB (which should be typed as a genuine TAB) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

Household Population		
Town	**Households**	
	Number	**Size**
Bedminster	**789**	**3.26**
Bernards Twp.	**3087**	**3.74**
Bernardsville	**2018**	**3.30**
Bound Brook	**3425**	**3.04**
Branchburg	**1644**	**3.49**
Bridgewater	**7897**	**3.81**
Far Hills	**240**	**3.19**

SEE ALSO

eqn(1), **nroff**(1), **troff**(1)

NAME

tcopy – copy a magnetic tape

SYNOPSIS

tcopy *source* [*destination*]

AVAILABILITY

SUNWesu

DESCRIPTION

tcopy copies the magnetic tape mounted on the tape drive specified by the *source* argument. The only assumption made about the contents of a tape is that there are two tape marks at the end.

When only a source drive is specified, **tcopy** scans the tape, and displays information about the sizes of records and tape files. If a destination is specified, **tcopy** makes a copies the source tape onto the *destination* tape, with blocking preserved. As it copies, **tcopy** produces the same output as it does when only scanning a tape.

SEE ALSO

mt(1), **ioctl**(2)

NOTES

tcopy will only run on systems supporting an associated set of **ioctl**(2) requests.

NAME

tee – replicate the standard output

SYNOPSIS

tee [–**a**] [–**i**] [*filename*] . . .

AVAILABILITY

SUNWcsu

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *filename*s.

OPTIONS

–**a**	Append the output to the *filename*s rather than overwriting them.
–**i**	Ignore interrupts.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tee** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tee** behaves.

LC_CTYPE

Determines how **tee** handles characters. When **LC_CTYPE** is set to a valid value, **tee** can display and handle text and filenames containing valid characters for that locale. **tee** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tee** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

environ(5)

NAME

telnet – user interface to a remote system using the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

AVAILABILITY

SUNWcsu

DESCRIPTION

telnet communicates with another host using the TELNET protocol. If **telnet** is invoked without arguments, it enters command mode, indicated by its prompt **telnet>**. In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see ‘‘Telnet Commands’’ below) with those arguments.

Once a connection has been opened, **telnet** enters input mode. In this mode, text typed is sent to the remote host. The input mode entered will be either “character at a time” or “line by line” depending on what the remote system supports.

In “character at a time” mode, most text typed is immediately sent to the remote host for processing.

In “line by line” mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The “local echo character” (initially **^E**) may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user’s **quit**, **intr**, and **flush** characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see **toggle**, **autoflush**, and **toggle**, **autosynch**) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of **quit** and **intr**).

While connected to a remote host, **telnet** command mode may be entered by typing the **telnet** escape character (initially **^]**). When in command mode, the normal terminal editing conventions are available.

USAGE

Telnet Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

open *host* [*port*]

Open a connection to the named host. If no port number is specified, **telnet** will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see **hosts**(4)) or an Internet address specified in the “dot notation” (see **inet**(7)).

close Close any open TELNET session and exit **telnet**. An EOF (in command mode) will also close a session and exit.

quit Same as **close**, above.

z Suspend **telnet**. This command only works when the user is using a shell that supports job control, such as **sh**(1).

mode *type*
type is either **line** (for “line by line” mode) or **character** (for “character at a time” mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

status Show the current status of **telnet**. This includes the peer one is connected to, as well as the current mode.

display [*argument*. . .]
Display all, or some, of the **set** and **toggle** values (see **toggle**, *arguments*).

? [*command*]
Get help. With no arguments, **telnet** print a help summary. If a command is specified, **telnet** will print the help information for just that command.

send *arguments*
Send one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

escape Send the current **telnet** escape character (initially **^]**).

synch Send the TELNET **SYNCH** sequence. This sequence discards all previously typed (but not yet read) input on the remote system. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system — if it does not work, a lower case “r” may be echoed on the terminal).

brk Send the TELNET **BRK** (Break) sequence, which may have significance to the remote system.

ip Send the TELNET **IP** (Interrupt Process) sequence, which aborts the currently running process on the remote system.

ao Send the TELNET **AO** (Abort Output) sequence, which flushes all output **from** the remote system **to** the user’s terminal.

ayt Send the TELNET **AYT** (Are You There) sequence, to which the remote system may or may not respond.

ec Send the TELNET **EC** (Erase Character) sequence, which erases the last character entered.

el Send the TELNET **EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga Send the TELNET **GA** (Go Ahead) sequence, which likely has no significance to the remote system.

nop Send the TELNET **NOP** (No Operation) sequence.

? Print out help information for the **send** command.

set *argument value*

Set any one of a number of **telnet** variables to a specific value. The special value “off” turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

echo This is the value (initially **ˆE**) which, when in “line by line“ mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for example, entering a password).

escape This is the **telnet** escape character (initially **ˆ]**) which enters **telnet** command mode (when connected to a remote system).

interrupt

If **telnet** is in **localchars** mode (see **toggle localchars**) and the **interrupt** character is typed, a TELNET **IP** sequence (see **send** and **ip**) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal’s **intr** character.

quit If **telnet** is in **localchars** mode (see **toggle localchars**) and the **quit** character is typed, a TELNET **BRK** sequence (see **send**, **brk**) is sent to the remote host. The initial value for the quit character is taken to be the terminal’s **quit** character.

flushoutput

If **telnet** is in **localchars** mode (see **toggle localchars**) and the **flushoutput** character is typed, a TELNET **AO** sequence (see **send**, **ao**) is sent to the remote host. The initial value for the flush character is taken to be the terminal’s **flush** character.

erase If **telnet** is in **localchars** mode (see **toggle localchars**), **and** if **telnet** is operating in “character at a time” mode, then when this character is typed, a TELNET **EC** sequence (see **send**, **ec**) is sent to the remote system. The initial value for the erase character is taken to be the terminal’s **erase** character.

kill If **telnet** is in **localchars** mode (see **toggle localchars**), **and** if **telnet** is operating in “character at a time” mode, then when this character is typed, a TELNET **EL** sequence (see **send**, **el**) is sent to the remote system. The initial value for the kill character is taken to be the terminal’s **kill** character.

eof If **telnet** is operating in “line by line” mode, entering this character as the first character on a line sends this character to the remote system. The initial value of the eof character is taken to be the terminal’s **eof** character.

toggle *arguments*...

Toggle (between TRUE and FALSE) various flags that control how **telnet** responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

autoflush

If **autoflush** and **localchars** are both TRUE, then when the **ao**, **intr**, or **quit** characters are recognized (and transformed into TELNET sequences; see **set** for details), **telnet** refuses to display any data on the user's terminal until the remote system acknowledges (using a TELNET **Timing Mark** option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh", otherwise FALSE (see **stty**(1)).

autosynch

If **autosynch** and **localchars** are both TRUE, then when either the **interrupt** or **quit** characters are typed (see **set** for descriptions of the **interrupt** and **quit** characters), the resulting TELNET sequence sent is followed by the TELNET **SYNCH** sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod Toggle RETURN mode. When this mode is enabled, most RETURN characters received from the remote host will be mapped into a RETURN followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends RETURN, but never LINEFEED. The initial value for this toggle is FALSE.

debug Toggle socket level debugging (useful only to the super-user). The initial value for this toggle is FALSE.

localchars

If this is TRUE, then the **flush**, **interrupt**, **quit**, **erase**, and **kill** characters (see **set**) are recognized locally, and transformed into appropriate TELNET control sequences (respectively **ao**, **ip**, **brk**, **ec**, and **el**; see **send**). The initial value for this toggle is TRUE in "line by line" mode, and FALSE in "character at a time" mode.

netdata

Toggle the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

options

Toggle the display of some internal **telnet** protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

? Display the legal **toggle** commands.

SEE ALSO

rlogin(1), **sh**(1), **stty**(1), **hosts**(4), **inet**(7)

NOTES

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in “line by line” mode.

There is enough settable state to justify a **file.**

In “line by line” mode, the terminal’s EOF character is only recognized (and sent to the remote system) when it is the first character on a line.

NAME

test – condition evaluation command

SYNOPSIS

/usr/ucb/test *expression*

[*expression*]

AVAILABILITY

SUNWscpu

DESCRIPTION

test evaluates the expression *expression* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; **test** also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second **SYNOPSIS** line) must be separate arguments to the **test** command; normally these items are separated by spaces.

USAGE

Primitives

The following primitives are used to construct *expression*:

–**r** *filename*	True if *filename* exists and is readable.
–**w** *filename*	True if *filename* exists and is writable.
–**x** *filename*	True if *filename* exists and is executable.
–**f** *filename*	True if *filename* exists and is a regular file. Alternatively, if **/usr/bin/sh** users specify **/usr/ucb** before **/usr/bin** in their PATH environment variable, then **test** will return true if *filename* exists and is (**not**–**a**–**directory**). This is also the default for **/usr/bin/csh** users.
–**d** *filename*	True if *filename* exists and is a directory.
–**c** *filename*	True if *filename* exists and is a character special file.
–**b** *filename*	True if *filename* exists and is a block special file.
–**p** *filename*	True if *filename* exists and is a named pipe (fifo).
–**u** *filename*	True if *filename* exists and its set-user-ID bit is set.
–**g** *filename*	True if *filename* exists and its set-group-ID bit is set.
–**k** *filename*	True if *filename* exists and its sticky bit is set.
–**s** *filename*	True if *filename* exists and has a size greater than zero.
–**t** [*fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–**z** *s1*	True if the length of string *s1* is zero.
–**n** *s1*	True if the length of the string *s1* is non-zero.
s1 **=** *s2*	True if strings *s1* and *s2* are identical.
s1 **!=** *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.

n1 –**eq** *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons –**ne**, –**gt**, –**ge**, –**lt**, and –**le** may be used in place of –**eq**.
–**L** *filename*	True if *filename* exists and is a symbolic link. With all other primitives, the symbolic links are followed by default.

Operators

These primaries may be combined with the following operators:

!	Unary negation operator.
–**a**	Binary *and* operator.
–**o**	Binary *or* operator (–**a** has higher precedence than –**o**).
(**expression**)	Parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), **sh**(1)

NOTES

The **not**–**a**–**directory** alternative to the –**f** option is a transition aid for BSD applications and may not be supported in future releases.

The –**L** option is a migration aid for users of other shells which have similar options and may not be supported in future releases.

If you test a file you own (the **-r ,** **-w ,** or **-x** tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The **=** and **!=** operators have a higher precedence than the –**r** through –**n** operators, and **=** and **!=** always expect arguments; therefore, **=** and **!=** cannot be used with the –**r** through –**n** operators.

If more than one argument follows the –**r** through –**n** operators, only the first argument is examined; the others are ignored, unless a –**a** or a –**o** is the second argument.

NAME

test – condition evaluation command

SYNOPSIS

test *expression*

[*expression*]

DESCRIPTION

test evaluates the expression *expression* and if its value is true, sets a zero (TRUE) exit status; otherwise, a non-zero (FALSE) exit status is set; **test** also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second **SYNOPSIS** line) must be separate arguments to **test**. Normally these items are separated by spaces.

USAGE

Primitives

The following primitives are used to construct *expression*:

–**r** *filename*	True if *filename* exists and is readable.
–**w** *filename*	True if *filename* exists and is writable.
–**x** *filename*	True if *filename* exists and is executable.
–**f** *filename*	True if *filename* exists and is a regular file.
–**d** *filename*	True if *filename* exists and is a directory.
–**c** *filename*	True if *filename* exists and is a character special file.
–**b** *filename*	True if *filename* exists and is a block special file.
–**p** *filename*	True if *filename* exists and is a named pipe (fifo).
–**u** *filename*	True if *filename* exists and its set-user-ID bit is set.
–**g** *filename*	True if *filename* exists and its set-group-ID bit is set.
–**k** *filename*	True if *filename* exists and its sticky bit is set.
–**s** *filename*	True if *filename* exists and has a size greater than zero.
–**t** [*fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–**z** *s1*	True if the length of string *s1* is zero.
–**n** *s1*	True if the length of the string *s1* is non-zero.
s1 **=** *s2*	True if strings *s1* and *s2* are identical.
s1 **!=** *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.

n1 –**eq** *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons –**ne**, –**gt**, –**ge**, –**lt**, and –**le** may be used in place of –**eq**.

Operators

These primaries may be combined with the following operators:

!	Unary negation operator.
–**a**	Binary *and* operator.
–**o**	Binary *or* operator (–**a** has higher precedence than –**o**).
`( *expression* )`	Parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), **sh**(1)

NOTES

If you test a file you own (the **-r , -w ,** or **-x** tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The **=** and **!=** operators have a higher precedence than the –**r** through –**n** operators, and **=** and **!=** always expect arguments; therefore, **=** and **!=** cannot be used with the –**r** through –**n** operators.

If more than one argument follows the –**r** through –**n** operators, only the first argument is examined; the others are ignored, unless a –**a** or a –**o** is the second argument.

NAME

tftp – trivial file transfer program

SYNOPSIS

tftp [*host*]

AVAILABILITY

SUNWcsu

DESCRIPTION

tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case **tftp** uses *host* as the default host for future transfers (see the **connect** command below).

USAGE

Once **tftp** is running, it issues the prompt **tftp>** and recognizes the following commands:

Commands

connect *host-name* **[** *port* **]**

Set the *host* (and optionally *port*) for transfers. The TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the **connect** command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the **connect** command; the remote host can be specified as part of the **get** or **put** commands.

mode *transfer-mode*

Set the mode for transfers; *transfer-mode* may be one of **ascii** or **binary**. The default is **ascii**.

put *filename*
put *localfile remotefile*
put *filename1 filename2 . . . filenameN remote-directory*

Transfer a file, or a set of files, to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form:

*host***:***filename*

to specify both a host and filename at the same time. If the latter form is used, the specified host becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be running the UNIX system. Files may be written only if they already exist and are publicly writable (see **in.tftpd**(1M)).

get *filename*
get *remotename localname*
get *filename1 filename2 filename3 . . . filenameN*

Get a file or set of files (three or more) from the specified remote *sources*. *source* can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form:

*host***:***filename*

to specify both a host and filename at the same time. If the latter form is used, the last host specified becomes the default for future transfers.

quit Exit **tftp**. An EOF also exits.

verbose
Toggle verbose mode.

trace Toggle packet tracing.

status Show current status.

rexmt *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

timeout *total-transmission-timeout*
Set the total transmission timeout, in seconds.

ascii Shorthand for **mode ascii**.

binary Shorthand for **mode binary**.

? [*command-name* **. . .]**
Print help information.

NOTES

The default *transfer-mode* is **ascii**. This differs from pre-SunOS 4.0 and pre-4.3BSD systems, so explicit action must be taken when transferring non-ASCII binary files such as executable commands.

Because there is no user-login or validation within the TFTP protocol, many remote sites restrict file access in various ways. Approved methods for file access are specific to each site, and therefore cannot be documented here.

When using the **get** command to transfer multiple files from a remote host, three or more files must be specified. If two files are specified, the second file is used as a local file.

NAME

time – time a command

SYNOPSIS

time *command*

AVAILABILITY

SUNWaccu

DESCRIPTION

The *command* is executed; after it is complete, **time** prints the elapsed time during the command (**real**), the time spent in the system (**sys**), and the time spent in execution of the *command* (**user**). Times are reported in seconds or in the format *hours:minutes:seconds* .

The times are printed on standard error.

EXAMPLES

The two examples here show the differences between the **csh** version of **time** and the version in **/usr/bin/time**. These examples assume that **csh** is the shell in use.

```
example% time find / -name csh.1 -print
/usr/share/man/man1/csh.1
95.0u 692.0s 1:17:52 16% 0+0k 0+0io 0pf+0w
```

For an explanation of the format of **time** output see **csh**(1).

```
example% /usr/bin/time find / -name csh.1 -print
/usr/share/man/man1/csh.1

real  1:23:31.5
user    1:33.2
sys    11:28.2
```

SEE ALSO

shell_builtins(1), **timex**(1), **times**(2)

NOTES

When the time command is run on a multiprocessor machine, the total of the values printed for **user** and **sys** can exceed **real**. This is because on a multiprocessor maching it is possible to divide the task between the various processors.

When the command being timed is interrupted, the timing values displayed may not always be accurate.

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 100th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

times – shell built-in function to report time usages of the current shell

SYNOPSIS

sh **times**

ksh † **times**

DESCRIPTION

sh Print the accumulated user and system times for processes run from the shell.

ksh Print the accumulated user and system times for the shell and for processes run from the shell.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

ksh(1), **sh**(1), **time**(1)

NAME

timex – time a command; report process data and system activity

SYNOPSIS

timex [**–o**] [**–p** [**–fhkmrt**]] [**–s**] *command*

AVAILABILITY

SUNWaccu

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of **timex** is written on standard error.

OPTIONS

–o Report the total number of blocks read or written and total characters transferred by *command* and all its children. This option works only if the process accounting software is installed.

–p List process accounting records for *command* and all its children. This option works only if the process accounting software is installed. Suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported. The options are as follows:

- **–f** Print the **fork**(2) / **exec**(2) flag and system exit status columns in the output.
- **–h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This ‘‘hog factor’’ is computed as (total CPU time) / (elapsed time).
- **–k** Instead of memory size, show total kcore-minutes.
- **–m** Show mean core size (the default).
- **–r** Show CPU factor (user time / (system-time + user-time).
- **–t** Show separate system and user CPU times. The number of blocks read or written and the number of characters transferred are always reported.

–s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in **sar**(1) are reported.

EXAMPLES

A simple example:

```
example% timex –ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
example% timex –opskmt sh

        session commands
EOT
```

SEE ALSO

sar(1), **time**(1), **times**(2)

NOTES

Process records associated with *command* are selected from the accounting file **/var/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user ID, terminal ID, and execution time window will be spuriously included.

NAME

tip – connect to remote system

SYNOPSIS

tip [–**v**] [–*speed-entry*] *hostname* | *phone-number*

AVAILABILITY

SUNWcsu

DESCRIPTION

tip establishes a full-duplex terminal connection to a remote host. Once the connection is established, a remote session using **tip** behaves like an interactive session on a local terminal.

The *remote* file contains entries describing remote systems and line speeds used by **tip**.

Each host has a default baud rate for the connection, or you can specify a speed with the –*speed-entry* command line argument.

When *phone-number* is specified, **tip** looks for an entry in the *remote* file of the form:

> **tip** –*speed-entry*

When it finds such an entry, it sets the connection speed accordingly. If it finds no such entry, **tip** interprets –*speed-entry* as if it were a system name, resulting in an error message.

If you omit –*speed-entry*, **tip** uses the **tip0** entry to set a speed for the connection.

When establishing the connection **tip** sends a connection message to the remote system. The default value for this message can be found in the *remote* file.

When **tip** attempts to connect to a remote system, it opens the associated device with an exclusive-open **ioctl**(2) call. Thus only one user at a time may access a device. This is to prevent multiple processes from sampling the terminal line. In addition, **tip** honors the locking protocol used by **uucp**(1C).

When **tip** starts up it reads commands from the file **.tiprc** in your home directory.

OPTIONS

–**v** Display commands from the **.tiprc** file as they are executed.

USAGE

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well).

At any time that **tip** prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, aborts the dialogue and returns you to the remote machine.

Commands

A tilde (**~**) appearing as the first character of a line is an escape signal which directs **tip** to perform some special action. **tip** recognizes the following escape sequences:

~^D
~. Drop the connection and exit (you may still be logged in on the remote machine).

~c [*name*]
Change directory to *name* (no argument implies change to your home directory).

˜! Escape to an interactive shell on the local machine (exiting the shell returns you to **tip**).

˜> Copy file from local to remote.

˜< Copy file from remote to local.

˜p *from* [*to*]

Send a file to a remote host running the UNIX system. When you use the put command, the remote system runs the command string

cat > *to*

while **tip** sends it the *from* file. If the *to* file is not specified, the *from* file name is used. This command is actually a UNIX-system-specific version of the '**˜>**' command.

˜t *from* [*to*]

Take a file from a remote host running the UNIX system. As in the put command the *to* file defaults to the *from* file name if it is not specified. The remote host executes the command string

cat *from* **; echo ˆA**

to send the file to **tip**.

˜| Pipe the output from a remote command to a local process. The command string sent to the local system is processed by the shell.

˜C Connect a program to the remote machine. The command string sent to the program is processed by the shell. The program inherits file descriptors 0 as remote line input, 1 as remote line output, and 2 as tty standard error.

˜$ Pipe the output from a local process to the remote host. The command string sent to the local system is processed by the shell.

˜# Send a BREAK to the remote system.

˜s Set a variable (see the discussion below).

˜ˆZ Stop **tip** (only available when run under a shell that supports job control, such as the C shell).

˜ˆY Stop only the "local side" of **tip** (only available when run under a shell that supports job control, such as the C shell); the "remote side" of **tip**, the side that displays output from the remote host, is left running.

˜? Get a summary of the tilde escapes.

Copying files requires some cooperation on the part of the remote host. When a **˜>** or **˜<** escape is used to send a file, **tip** prompts for a file name (to be transmitted or received) and a command to be sent to the remote system, in case the file is being transferred from the remote system. While **tip** is transferring a file the number of lines transferred will be continuously displayed on the screen. A file transfer may be aborted with an interrupt.

Auto-call Units

tip may be used to dial up remote systems using a number of auto-call unit's (ACU's). When the remote system description contains the **du** capability, **tip** uses the call-unit (**cu**), ACU type (**at**), and phone numbers (**pn**) supplied. Normally **tip** displays verbose messages as it dials.

Depending on the type of auto-dialer being used to establish a connection the remote host may have garbage characters sent to it upon connection. The user should never assume that the first characters typed to the foreign host are the first ones presented to it. The recommended practice is to immediately type a **kill** character upon establishing a connection (most UNIX systems either support **@** or CTRL-U as the initial kill character).

tip currently supports the Ventel MD-212+ modem and DC Hayes-compatible modems.

When **tip** initializes a Hayes-compatible modem for dialing, it sets up the modem to auto-answer. Normally, after the conversation is complete, **tip** drops DTR, which causes the modem to "hang up."

Most modems can be configured such that when DTR drops, they re-initialize themselves to a preprogrammed state. This can be used to reset the modem and disable auto-answer, if desired.

Additionally, it is possible to start the phone number with a Hayes **S** command so that you can configure the modem before dialing. For example, to disable auto-answer, set up all the phone numbers in **/etc/remote** using something like **pn=S0=0DT5551212**. The **S0=0** disables auto-answer.

Remote Host Description

Descriptions of remote hosts are normally located in the system-wide file **/etc/remote**. However, a user may maintain personal description files (and phone numbers) by defining and exporting the REMOTE shell variable. The *remote* file must be readable by **tip**, but a secondary file describing phone numbers may be maintained readable only by the user. This secondary phone number file is **/etc/phones**, unless the shell variable PHONES is defined and exported. The phone number file contains lines of the form:

system-name phone-number

Each phone number found for a system is tried until either a connection is established, or an end of file is reached. Phone numbers are constructed from '**0123456789**–=*', where the '=' and '*' are used to indicate a second dial tone should be waited for (ACU dependent).

tip Internal Variables

tip maintains a set of variables which are used in normal operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the **~s** escape. The syntax for variables is patterned after **vi**(1) and **mail**(1). Supplying **all** as an argument to the **~s** escape displays all variables that the user can read. Alternatively, the user may request display of a particular variable by attaching a **?** to the end. For example '**~s escape?**' displays the current escape character.

Variables are numeric (num), string (str), character (char), or Boolean (bool) values. Boolean variables are set merely by specifying their name. They may be reset by prepending a **!** to the name. Other variable types are set by appending an **=** and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables.

Variables may be initialized at run time by placing set commands (without the **~s** prefix) in a **.tiprc** file in one's home directory. The **–v** option makes **tip** display the sets as they are made. Comments preceded by a **#** sign can appear in the **.tiprc** file.

Finally, the variable names must either be completely specified or an abbreviation may be given. The following list details those variables known to **tip**.

beautify

(bool) Discard unprintable characters when a session is being scripted; abbreviated **be**. If the **nb** capability is present, **beautify** is initially set to **off**; otherwise, **beautify** is initially set to **on**.

baudrate

(num) The baud rate at which the connection was established; abbreviated **ba**. If a baud rate was specified on the command line, **baudrate** is initially set to the specified value; otherwise, if the **br** capability is present, **baudrate** is initially set to the value of that capability; otherwise, **baudrate** is set to 300 baud. Once **tip** has been started, **baudrate** can only changed by the super-user.

dialtimeout

(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated **dial**. **dialtimeout** is initially set to 60 seconds, and can only changed by the super-user.

disconnect

(str) The string to send to the remote host to disconnect from it; abbreviated **di**. If the **di** capability is present, **disconnect** is initially set to the value of that capability; otherwise, **disconnect** is set to a null string ("").

echocheck

(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; abbreviated **ec**. If the **ec** capability is present, **echocheck** is initially set to **on**; otherwise, **echocheck** is initially set to **off**.

eofread

(str) The set of characters which signify an end-of-transmission during a **~<** file transfer command; abbreviated **eofr**. If the **ie** capability is present, **eofread** is initially set to the value of that capability; otherwise, **eofread** is set to a null string ("").

eofwrite

(str) The string sent to indicate end-of-transmission during a **~>** file transfer command; abbreviated **eofw**. If the **oe** capability is present, **eofread** is initially set to the value of that capability; otherwise, **eofread** is set to a null string ("").

eol	(str) The set of characters which indicate an end-of-line. **tip** will recognize escape characters only after an end-of-line. If the **el** capability is present, **eol** is initially set to the value of that capability; otherwise, **eol** is set to a null string ("").
escape	(char) The command prefix (escape) character; abbreviated **es**. If the **es** capability is present, **escape** is initially set to the value of that capability; otherwise, **escape** is set to ‘**~**’.
etimeout	(num) The amount of time, in seconds, that **tip** should wait for the echo-check response when **echocheck** is set; abbreviated **et**. If the **et** capability is present, **etimeout** is initially set to the value of that capability; otherwise, **etimeout** is set to 10 seconds.
exceptions	(str) The set of characters which should not be discarded due to the beautification switch; abbreviated **ex**. If the **ex** capability is present, **exceptions** is initially set to the value of that capability; otherwise, **exceptions** is set to ‘**\t\n\f\b**’.
force	(char) The character used to force literal data transmission; abbreviated **fo**. If the **fo** capability is present, **force** is initially set to the value of that capability; otherwise, **force** is set to **\377** (which disables it).
framesize	(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated **fr**. If the **fs** capability is present, **framesize** is initially set to the value of that capability; otherwise, **framesize** is set to 1024.
halfduplex	(bool) Do local echoing because the host is half-duplex; abbreviated **hdx**. If the **hd** capability is present, **halfduplex** is initially set to **on**; otherwise, **halfduplex** is initially set to **off**.
hardwareflow	(bool) Do hardware flow control; abbreviated **hf**. If the **hf** capability is present, **hardwareflow** is initially set to **on**; otherwise, **hardwareflowcontrol** is initially set to **off**.
host	(str) The name of the host to which you are connected; abbreviated **ho**. **host** is permanently set to the name given on the command line or in the HOST environment variable.
localecho	(bool) A synonym for **halfduplex**; abbreviated **le**.
log	(str) The name of the file to which to log information about outgoing phone calls. **log** is initially set to **/var/adm/aculog**, and can only be inspected or changed by the super-user.

parity (str) The parity to be generated and checked when talking to the remote host; abbreviated **par**. The possible values are:

none	
zero	Parity is not checked on input, and the parity bit is set to zero on output.
one	Parity is not checked on input, and the parity bit is set to one on output.
even	Even parity is checked for on input and generated on output.
odd	Odd parity is checked for on input and generated on output.

If the **pa** capability is present, **parity** is initially set to the value of that capability; otherwise, **parity** is set to **none**.

phones
The file in which to find hidden phone numbers. If the environment variable PHONES is set, **phones** is set to the value of PHONES; otherwise, **phones** is set to **/etc/phones**. The value of **phones** cannot be changed from within **tip**.

prompt
(char) The character which indicates an end-of-line on the remote host; abbreviated **pr**. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on receipt of this character. If the **pr** capability is present, **prompt** is initially set to the value of that capability; otherwise, **prompt** is set to **\n**.

raise (bool) Upper case mapping mode; abbreviated **ra**. When this mode is enabled, all lower case letters will be mapped to upper case by **tip** for transmission to the remote machine. If the **ra** capability is present, **raise** is initially set to **on**; otherwise, **raise** is initially set to **off**.

raisechar
(char) The input character used to toggle upper case mapping mode; abbreviated **rc**. If the **rc** capability is present, **raisechar** is initially set to the value of that capability; otherwise, **raisechar** is set to **\377** (which disables it).

rawftp (bool) Send all characters during file transfers; do not filter non-printable characters, and do not do translations like **\n** to **\r**. Abbreviated **raw**. If the **rw** capability is present, **rawftp** is initially set to **on**; otherwise, **rawftp** is initially set to **off**.

record (str) The name of the file in which a session script is recorded; abbreviated **rec**. If the **re** capability is present, **record** is initially set to the value of that capability; otherwise, **record** is set to **tip.record**.

remote The file in which to find descriptions of remote systems. If the environment variable REMOTE is set, **remote** is set to the value of REMOTE; otherwise, **remote** is set to **/etc/remote**. The value of **remote** cannot be changed from within **tip**.

script (bool) Session scripting mode; abbreviated **sc**. When **script** is **on**, **tip** will record everything transmitted by the remote machine in the script record file specified in **record**. If the **beautify** switch is on, only printable ASCII characters will be included in the script file (those characters between 040 and 0177). The variable **exceptions** is used to indicate characters which are an exception to the normal beautification rules. If the **sc** capability is present, **script** is initially set to **on**; otherwise, **script** is initially set to **off**.

tabexpand
(bool) Expand TAB characters to SPACE characters during file transfers; abbreviated **tab**. When **tabexpand** is **on**, each tab is expanded to 8 SPACE characters. If the **tb** capability is present, **tabexpand** is initially set to **on**; otherwise, **tabexpand** is initially set to **off**.

tandem
(bool) Use XON / XOFF flow control to limit the rate that data is sent by the remote host; abbreviated **ta**. If the **nt** capability is present, **tandem** is initially set to **off**; otherwise, **tandem** is initially set to **on**.

verbose
(bool) Verbose mode; abbreviated **verb**; When verbose mode is enabled, **tip** prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more. If the **nv** capability is present, **verbose** is initially set to **off**; otherwise, **verbose** is initially set to **on**.

SHELL (str) The name of the shell to use for the **~!** command; default value is **/bin/sh**, or taken from the environment.

HOME (str) The home directory to use for the **~c** command; default value is taken from the environment.

EXAMPLES

An example of the dialogue used to transfer files is given below.

```
arpa% tip monet
[connected]
...(assume we are talking to a UNIX system)...
ucbmonet login: sam
Password:
monet% cat > sylvester.c
~> Filename: sylvester.c
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~< Filename: reply.c
List command for remote host: cat reply.c
65 lines transferred in 2 minutes
monet%
...(or, equivalently)...
monet% ~p sylvester.c
...(actually echoes as ~[put] sylvester.c)...
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~t reply.c
...(actually echoes as ~[take] reply.c)...
```

```
65 lines transferred in 2 minutes
monet%
...(to print a file locally)...
monet% ˜|Local command: pr −h sylvester.c | lpr
List command for remote host: cat sylvester.c
monet% ˜ˆD
[EOT]
...(back on the local system)...
```

ENVIRONMENT

The following environment variables are read by **tip**.

REMOTE	The location of the *remote* file.
PHONES	The location of the file containing private phone numbers.
HOST	A default host to connect to.
HOME	One's log-in directory (for chdirs).
SHELL	The shell to fork on a '**˜!**' escape.

FILES

/etc/phones	
/etc/remote	
/var/spool/locks/LCK ..*	lock file to avoid conflicts with UUCP
/var/adm/aculog	file in which outgoing calls are logged
˜/.tiprc	initialization file

SEE ALSO

cu(1C), **mail**(1), **uucp**(1C), **vi**(1), **ioctl**(2)

BUGS

There are two additional variables **chardelay** and **linedelay** that are currently not implemented.

NAME

touch – update access time and / or modification time of a file

SYNOPSIS

touch [–**amc**] [*mmddhhmm*[*yy*]] *filename* . . .

AVAILABILITY

SUNWcsu

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see **date**(1)) the current time is used.

The return code from **touch** is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

OPTIONS

–**a**	**touch** updates only the access time respectively (default is –**am**).
–**m**	**touch** updates only the modification time.
–**c**	Silently prevents **touch** from creating the file if it did not previously exist.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **touch** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **touch** behaves.

LC_CTYPE

Determines how **touch** handles characters. When **LC_CTYPE** is set to a valid value, **touch** can display and handle text and filenames containing valid characters for that locale. **touch** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **touch** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

date(1), **utime**(2), **environ**(5)

NOTES

Users familiar with the BSD environment will find that the –**f** option is accepted, but ignored. The –**f** option is unnecessary since **touch** will succeed for all files owned by the user regardless of the permissions on the files.

NAME

tput – initialize a terminal or query terminfo database

SYNOPSIS

tput [–**T***type*] *capname* [*parms* . . .]
tput [–**T***type*] **init**
tput [–**T***type*] **reset**
tput [–**T***type*] **longname**
tput –**S** **<<**

AVAILABILITY

SUNWcsu

DESCRIPTION

tput uses the **terminfo** database to make the values of terminal-dependent capabilities and information available to the shell (see **sh**(1)), to initialize or reset the terminal, or return the long name of the requested terminal type. **tput** outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, **tput** simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code (**$?**, see **sh**(1)) to be sure it is **0**. (See the **EXIT CODES** and **DIAGNOSTICS** sections.) For a complete list of capabilities and the *capname* associated with each, see **terminfo**(4).

OPTIONS

–**T***type*	Indicates the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable **TERM**. If –**T** is specified, then the shell variables **LINES** and **COLUMNS** and the layer size will not be referenced.
capname	Indicates the attribute from the **terminfo** database.
parms	If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.
–**S**	Allows more than one capability per invocation of **tput**. The capabilities must be passed to **tput** from the standard input instead of from the command line (see the example in the **EXAMPLES** section). Only one *capname* is allowed per line. The –**S** option changes the meaning of the **0** and **1** boolean and string exit codes (see the **EXIT CODES** section).
init	If the **terminfo** database is present and an entry for the user's terminal exists (see –**T***type*, above), the following will occur: (1) if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (for instance, newline) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and

(4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname If the **terminfo** database is present and an entry for the user's terminal exists (see –**T***type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the **terminfo** database (see **term**(5)).

EXIT CODES

If *capname* is of type boolean, a value of **0** is set for TRUE and **1** for FALSE unless the –**S** option is used.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type boolean or string and the –**S** option is used, a value of **0** is returned to indicate that all lines were successful. No indication of which line failed can be given so exit code **1** will never appear. Exit codes **2**, **3**, and **4** retain their usual interpretation.

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of –**1** means that *capname* is not defined for this terminal *type*.

Any other exit code indicates an error; see the **DIAGNOSTICS** section.

EXAMPLES

This example initializes the terminal according to the type of terminal in the environment variable **TERM**. This command should be included in everyone's .profile after the environment variable **TERM** has been exported, as illustrated on the **profile**(4) manual page.

```
example% tput init
```

The next example resets an AT&T 5620 terminal, overriding the type of terminal in the environment variable **TERM**.

```
example% tput –T5620 reset
```

The following example sends the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position).

```
example% tput cup 0 0
```

The next example echos the clear-screen sequence for the current terminal.

```
example% tput clear
```

The next command prints the number of columns for the current terminal.

```
example% tput cols
```

The following command prints the number of columns for the 450 terminal.

```
example% tput –T450 cols
```

The next example sets the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt:

```
echo "${bold}Please type in your name: ${offbold}\c"

example% bold=‘tput smso‘
example% offbold=‘tput rmso‘
```

This example sets the exit code to indicate if the current terminal is a hardcopy terminal.

```
example% tput hc
```

This next example sends the sequence to move the cursor to row **23,** column **4.**

```
example% tput cup 23 4
```

The next command prints the long name from the **terminfo** database for the type of terminal specified in the environment variable **TERM**.

```
example% tput longname
```

This last example shows tput processing several capabilities in one invocation. This example clears the screen, moves the cursor to position 10, 10 and turns on bold (extra bright) mode. The list is terminated by an exclamation mark (**!**) on a line by itself.

```
example% tput –S <<!
> clear
> cup 10 10
> bold
> !
```

FILES

/usr/include/curses.h	**curses**(3X) header
/usr/include/term.h	**terminfo** header
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of **terminfo**(4)
/usr/share/lib/terminfo/?/*	compiled terminal description database

SEE ALSO

clear(1), **stty**(1), **tabs**(1), **profile**(4), **terminfo**(4)

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

exit code	error message
0	−1 (*capname* is a numeric variable that is not specified in the **terminfo** database for this terminal type, for instance, **tput −T450 lines** and **tput −T2621 xmc**).
1	no error message is printed, see the **EXIT CODES** section.
2	usage error.
3	unknown terminal *type* or no **terminfo** database.
4	unknown **terminfo** capability *capname*.

NAME

tr – translate characters

SYNOPSIS

tr [–**cds**] [*string1* [*string2*]]

AVAILABILITY

SUNWcsu

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*.

When *string2* (with any repetitions of characters) contains fewer characters than *string1*, characters in *string1* with no corresponding character in *string2* are not translated.

OPTIONS

Any combination of the options –**cds** may be used:

–**c**	Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
–**d**	Deletes all input characters in *string1*.
–**s**	Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a–z]	Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.
[a**n*]**	Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLES

The following example creates a list of all the words in *filename1* one per line in *filename2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
example% tr –cs "[A–Z][a–z]" "[\012*]" <filename1> filename2
```

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tr** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tr** behaves.

LC_CTYPE

Determines how **tr** handles characters. When **LC_CTYPE** is set to a valid value, **tr** can display and handle text and filenames containing valid characters for that locale. **tr** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tr** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

ed(1), **sh**(1), **ascii**(5), **environ**(5)

NOTES

Will not handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

NAME

tr – translate characters

SYNOPSIS

/usr/ucb/tr [**–cds**] [*string1* [*string2*]]

AVAILABILITY

SUNWscpu

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. The arguments *string1* and *string2* are considered sets of characters. Any input character found in *string1* is mapped into the character in the corresponding position within *string2*. When *string2* is short, it is padded to the length of *string1* by duplicating its last character.

In either string the notation:

a–b

denotes a range of characters from *a* to *b* in increasing ASCII order. The character \ , followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. As with the shell, the escape character \ , followed by any other character, escapes any special meaning for that character.

OPTIONS

Any combination of the options –**c**, –**d**, or –**s** may be used:

–**c**	Complement the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal.
–**d**	Delete all input characters in *string1*.
–**s**	Squeeze all strings of repeated output characters that are in *string2* to single characters.

EXAMPLES

The following example creates a list of all the words in *filename1* one per line in *filename2*, where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect ‘ \ ’ from the shell. 012 is the ASCII code for NEWLINE.

```
example% tr –cs A–Za–z ´\012´ <filename1> filename2
```

SEE ALSO

ed(1), **ascii**(5)

NOTES

Will not handle ASCII NUL in *string1* or *string2*. **tr** always deletes NUL from input.

NAME

trap, onintr – shell built-in functions to respond to (hardware) signals

SYNOPSIS

sh

trap [*argument*] [*n*] . . .

csh

onintr [– | *label*]

ksh

† **trap** [*arg*] [*sig*] . . .

DESCRIPTION

sh

The **trap** command *argument* is to be read and executed when the shell receives numeric or symbolic signal(s) (*n*). (Note: *argument* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number or corresponding symbolic names. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *argument* is absent all trap(s) *n* are reset to their original values. If *argument* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *argument* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

csh

onintr controls the action of the shell on interrupts. With no arguments, **onintr** restores the default action of the shell on interrupts. (The shell terminates shell scripts and returns to the terminal command input level). With the – argument, the shell ignores all interrupts. With a *label* argument, the shell executes a **goto** *label* when an interrupt is received or a child process terminates because it was interrupted.

ksh

trap uses *arg* as a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. **trap** commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is omitted or is –, then the trap(s) for each *sig* are reset to their original values. If *arg* is the NULL (the empty string, e.g., "") string then this signal is ignored by the shell and by the commands it invokes. If *sig* is **ERR** then *arg* will be executed whenever a command has a non-zero exit status. If *sig* is **DEBUG** then *arg* will be executed after each command. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

csh(1), **exit**(1), **ksh**(1), **sh**(1)

NAME

troff – typeset or format documents

SYNOPSIS

troff [–**a**] [–**f**] [–**F***dir*] [–**i**] [–**m***name*] [–**n***N*] [–**o***list*] [–**r***aN*] [–**s***N*] [–**T***dest*]
[–**u***N*] [–**z**] [*filename*] . . .

AVAILABILITY

SUNWdoc

DESCRIPTION

troff formats text in the *filename*s for typesetting or laser printing. Input to **troff** is expected to consist of text interspersed with formatting requests and macros. If no *filename* argument is present, **troff** reads standard input. A minus sign (–) as a *filename* indicates that standard input should be read at that point in the list of input files.

The following options may appear in any order, but all must appear before the first *filename*.

–**a**	Send an ASCII approximation of formatted output to standard output.
–**f**	Do not print a trailer after the final page of output or cause the postprocessor to relinquish control of the device.
–**F***dir*	Search directory *dir* for font width or terminal tables instead of the system default directory.
–**i**	Read standard input after all input files are exhausted.
–**m***name*	Prepend the macro file **/usr/share/lib/tmac/***name* to the input *filename*s. Note: most references to macro packages include the leading *m* as part of the name; for example, the **man**(5) macros reside in **/usr/share/lib/tmac/an**. The macro directory can be changed by setting the **TROFFMACS** environment variable to a specific path. Be certain to include the trailing ' / ' (slash) at the end of the path.
–**n***N*	Number the first generated page *N*.
–**o***list*	Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end.
–**q**	Quiet mode in **nroff**; ignored in **troff**.
–**r***aN*	Set register *a* (one-character names only) to *N*.
–**s***N*	Stop the phototypesetter every *N* pages. On some devices, **troff** produces a trailer so you can change cassettes; resume by pressing the typesetter's start button.
–**T***dest*	Prepare output for typesetter *dest*. The following values can be supplied for *dest*: **post** A PostScript printer; this is the default value. **aps** Autologic APS-5.
–**u***N*	Set the emboldening factor for the font mounted in position 3 to *N*. If *N* is missing, then set the emboldening factor to 0.
–**z**	Suppress formatted output. Only diagnostic messages and messages output using the **.tm** request are output.

FILES

/tmp/trtmp	temporary file
/usr/share/lib/tmac/∗	standard macro files
/usr/lib/font/∗	font width tables for alternate mounted **troff** fonts
/usr/share/lib/nterm/∗	terminal driving tables for **nroff**

SEE ALSO

checknr(1), **col**(1), **dpost**(1), **eqn**(1), **lp**(1), **man**(1), **nroff**(1), **tbl**(1), **man**(5), **me**(5), **ms**(5)

NOTES

troff is not 8-bit clean because it is by design based on 7-bit ASCII.

NAME

true, false – provide truth values

SYNOPSIS

true

false

AVAILABILITY

SUNWcsu

DESCRIPTION

true does nothing, successfully. **false** does nothing, unsuccessfully. They are typically used in a shell script **sh** as:

```
while true
do
        command
done
```

SEE ALSO

sh(1)

DIAGNOSTICS

true has exit status zero.

false has non-zero exit status.

NAME

truss – trace system calls and signals

SYNOPSIS

truss [–**fcaeil**] [–[**tvx**] [**!**]*syscall*...] [–**s** [**!**]*signal*...] [–**m** [**!**]*fault*...] [–[**rw**] [**!**]*fd*...]
[–**o** *outfile*] *command* | –**p** *pid*

DESCRIPTION

truss executes the specified command and produces a trace of the system calls it performs, the signals it receives, and the machine faults it incurs. Each line of the trace output reports either the fault or signal name or the system call name with its arguments and return value(s). System call arguments are displayed symbolically when possible using defines from relevant system headers; for any pathname pointer argument, the pointed-to string is displayed. Error returns are reported using the error code names described in **intro**(2).

OPTIONS

The following options are recognized. For those options that take a list argument, the name **all** can be used as a shorthand to specify all possible members of the list. If the list begins with a **!**, the meaning of the option is negated (for example, exclude rather than trace). Multiple occurrences of the same option may be specified. For the same name in a list, subsequent options (those to the right) override previous ones (those to the left).

–p Interpret the *command* arguments to **truss** as a list of process-ids for existing processes (see **ps**(1)) rather than as a command to be executed. **truss** takes control of each process and begins tracing it provided that the userid and groupid of the process match those of the user or that the user is a privileged user. Processes may also be specified by their names in the **/proc** directory, for example, **/proc/12345**.

–f Follow all children created by **fork**() or **vfork**() and include their signals, faults, and system calls in the trace output. Normally, only the first-level command or process is traced. When –**f** is specified, the process-id is included with each line of trace output to indicate which process executed the system call or received the signal.

–c Count traced system calls, faults, and signals rather than displaying the trace line-by-line. A summary report is produced after the traced command terminates or when **truss** is interrupted. If –**f** is also specified, the counts include all traced system calls, faults, and signals for child processes.

–a Show the argument strings that are passed in each **exec**() system call.

–e Show the environment strings that are passed in each **exec**() system call.

–i Do not display interruptible sleeping system calls. Certain system calls, such as **open**() and **read**() on terminal devices or pipes can sleep for indefinite periods and are interruptible. Normally, **truss** reports such sleeping system calls if they remain asleep for more than one second. The system call is reported again a second time when it completes. The –**i** option causes such system calls to be reported only once, when they complete.

–l	Include the id of the responsible lightweight process with each line of trace output. If **–f** is also specified, both the process-id and the lightweight process id are included.
–t [**!**]*syscall,...*	System calls to trace or exclude. Those system calls specified in the comma-separated list are traced. If the list begins with a **!**, the specified system calls are excluded from the trace output. Default is –**tall**.
–v [**!**]*syscall,...*	Verbose. Display the contents of any structures passed by address to the specified system calls (if traced). Input values as well as values returned by the operating system are shown. For any field used as both input and output, only the output value is shown. Default is –**v!all**.
–x [**!**]*syscall,...*	Display the arguments to the specified system calls (if traced) in raw form, usually hexadecimal, rather than symbolically. This is for unredeemed hackers who must see the raw bits to be happy. Default is –**x!all**.
–s [**!**]*signal,...*	Signals to trace or exclude. Those signals specified in the comma-separated list are traced. The trace output reports the receipt of each specified signal, even if the signal is being ignored (not blocked). (Blocked signals are not received until they are unblocked.) Signals may be specified by name or number (see **<sys/signal.h>**). If the list begins with a **!**, the specified signals are excluded from the trace output. Default is –**sall**.
–m [**!**]*fault,...*	Machine faults to trace or exclude. Those machine faults specified in the comma-separated list are traced. Faults may be specified by name or number (see **<sys/fault.h>**). If the list begins with a **!**, the specified faults are excluded from the trace output. Default is –**mall** –**m!fltpage**.
–r [**!**]*fd,...*	Show the full contents of the I/O buffer for each **read**() on any of the specified file descriptors. The output is formatted 32 bytes per line and shows each byte as an ascii character (preceded by one blank) or as a 2-character C language escape sequence for control characters such as horizontal tab (\ t) and newline (\ n). If ascii interpretation is not possible, the byte is shown in 2-character hexadecimal representation. (The first 12 bytes of the I/O buffer for each traced **read**() are shown even in the absence of –**r**.) Default is –**r!all**.
–w [**!**]*fd,...*	Show the contents of the I/O buffer for each **write**() on any of the specified file descriptors (see –**r**). Default is –**w!all**.
–o *outfile*	File to be used for the trace output. By default, the output goes to standard error.

See Section 2 of the *man Pages(2): System Calls* for system call names accepted by the –**t**, –**v**, and –**x** options. System call numbers are also accepted.

If **truss** is used to initiate and trace a specified command and if the **–o** option is used or if standard error is redirected to a non-terminal file, then **truss** runs with hangup, interrupt, and quit signals ignored. This facilitates tracing of interactive programs that catch interrupt and quit signals from the terminal.

If the trace output remains directed to the terminal, or if existing processes are traced (the **–p** option), then **truss** responds to hangup, interrupt, and quit signals by releasing all traced processes and exiting. This enables the user to terminate excessive trace output and to release previously-existing processes. Released processes continue normally, as though they had never been touched.

EXAMPLES

This example produces a trace of the **find**(1) command on the terminal:

```
example% truss find . –print >find.out
```

Or, to see only a trace of the open, close, read, and write system calls:

```
example% truss –t open,close,read,write find . –print >find.out
```

This produces a trace of the **spell**(1) command on the file **truss.out**:

```
example% truss –f –o truss.out spell document
```

spell is a shell script, so the **–f** flag is needed to trace not only the shell but also the processes created by the shell. (The spell script runs a pipeline of eight concurrent processes.)

A particularly boring example is:

```
example% truss nroff –mm document >nroff.out
```

because 97% of the output reports **lseek**(), **read**(), and **write**() system calls. To abbreviate it:

```
example% truss –t !lseek,read,write nroff –mm document >nroff.out
```

This example verbosely traces the activity of process #1, **init**(1M) (if you are a privileged user):

```
example% truss –p –v all 1
```

Interrupting **truss** returns **init** to normal operation.

FILES

/proc/*nnnnn*	process files
/proc/process-id	

SEE ALSO

intro(2), **proc**(4)

NOTES

Some of the system calls described in Section 2 differ from the actual operating system interfaces. Do not be surprised by minor deviations of the trace output from the descriptions in Section 2.

Every machine fault (except a page fault) results in the posting of a signal to the lightweight process that incurred the fault. A report of a received signal will immediately follow each report of a machine fault (except a page fault) unless that signal is being blocked.

The operating system enforces certain security restrictions on the tracing of processes. In particular, any command whose object file (**a.out**) cannot be read by a user cannot be traced by that user; set-uid and set-gid commands can be traced only by a privileged user. Unless it is run by a privileged user, **truss** loses control of any process that performs an **exec**() of a set-id or unreadable object file; such processes continue normally, though independently of **truss**, from the point of the **exec**().

To avoid collisions with other controlling processes, **truss** will not trace a process that it detects is being controlled by another process via the **/proc** interface. This allows **truss** to be applied to **proc**(4)-BASED debuggers as well as to another instance of itself.

The trace output contains tab characters under the assumption that standard tab stops are set (every eight positions).

The trace output for multiple processes or for a multithreaded process (one that contains more than one lightweight process) is not produced in strict time order. For example, a **read**() on a pipe may be reported before the corresponding **write**(). For any one lightweight process (a traditional process contains only one), the output is strictly time-ordered.

The system may run out of per-user process slots when tracing of children is requested. When tracing more than one process, **truss** runs as one controlling process for each process being traced. For the example of the **spell** command shown above, **spell** itself uses 9 process slots, one for the shell and 8 for the 8-member pipeline, while **truss** adds another 9 processes, for a total of 18. This is perilously close to the usual system-imposed limit of 25 processes per user.

Not all possible structures passed in all possible system calls are displayed under the **–v** option.

NAME

tset, reset – establish or restore terminal characteristics

SYNOPSIS

tset [–**InQrs**] [–**e***c*] [–**k***c*] [–**m** [*port –ID* [*baudrate*] : *type*] . . .] [*type*]

reset [–] [–**e***c*] [–**I**] [–**k***c*] [–**n**] [–**Q**] [–**r**] [–**s**]
[–**m** [*indent*] [*test baudrate*]: *type*] . . . [*type*]

AVAILABILITY

SUNWscpu

DESCRIPTION

tset sets up your terminal, typically when you first log in. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. **tset** first determines the *type* of terminal involved, and then does necessary initializations and mode settings. If a port is not wired permanently to a specific terminal (not hardwired) it is given an appropriate generic identifier such as **dialup**.

reset clears the terminal settings by turning off CBREAK and RAW modes, output delays and parity checking, turns on NEWLINE translation, echo and TAB expansion, and restores undefined special characters to their default state. It then sets the modes as usual, based on the terminal type (which will probably override some of the above). See **stty**(1) for more information. All arguments to **tset** may be used with **reset**. **reset** also uses **rs=** and **rf=** to reset the initialization string and file. This is useful after a program dies and leaves the terminal in a funny state. Often in this situation, characters will not echo as you type them. You may have to type '<LINEFEED>**reset**<LINEFEED>' since '<RETURN>' may not work.

When no arguments are specified, **tset** reads the terminal type from the **TERM** environment variable and re-initializes the terminal, and performs initialization of mode, environment and other options at login time to determine the terminal type and set up terminal modes.

When used in a startup script (**.profile** for **sh**(1) users or **.login** for **csh**(1) users) it is desirable to give information about the type of terminal you will usually use on ports that are not hardwired. Any of the alternate generic names given in **/usr/share/lib/terminfo/*** may be used for the identifier. Refer to the –**m** option below for more information. If no mapping applies and a final *type* option, not preceded by a –**m**, is given on the command line then that type is used.

It is usually desirable to return the terminal type, as finally determined by **tset**, and information about the terminal's capabilities, to a shell's environment. This can be done using the –, –**s**, or –**S** options.

For the Bourne shell, put this command in your **.profile** file:

```
eval `tset –s options...`
```

or using the C shell, put these commands in your **.login** file:

```
set noglob
eval `tset –s options...`
unset noglob
```

With the C shell, it is also convenient to make an alias in your **.cshrc** file:

```
alias ts ´eval `tset –s \!*`´
```

This also allows the command:

```
ts 2621
```

to be invoked at any time to set the terminal and environment. It is not possible to get this aliasing effect with a Bourne shell script, because shell scripts cannot set the environment of their parent. If a process could set its parent's environment, none of this nonsense would be necessary in the first place.

Once the terminal type is known, **tset** sets the terminal driver mode. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. TAB and NEWLINE expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is '**#**', the erase character is changed as if **-e** had been used.

OPTIONS

–	The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the **TERM** environment variable.
–**e***c*	Set the erase character to be the named character *c* on all terminals. Default is the BACKSPACE key on the keyboard, usually **ˆH** (CTRL-H). The character *c* can either be typed directly, or entered using the circumflex-character notation used here.
–**i***c*	Set the interrupt character to be the named character *c* on all terminals. Default is **ˆC** (CTRL-C). The character *c* can either be typed directly, or entered using the circumflex-character notation used here.
–**I**	Suppress transmitting terminal-initialization strings.
–**k***c*	Set the line kill character to be the named character *c* on all terminals. Default is **ˆU** (CTRL-U). The kill character is left alone if –**k** is not specified. Control characters can be specified by prefixing the alphabetical character with a circumflex (as in CTRL-U) instead of entering the actual control key itself. This allows you to specify control keys that are currently assigned.
–**n**	Specify that the new tty driver modes should be initialized for this terminal. Probably useless since **stty new** is the default.

–Q Suppress printing the ‘**Erase set to**’ and ‘**Kill set to**’ messages.

–r In addition to other actions, reports the terminal type.

–s Output commands to set and export **TERM**. This can be used with

```
set noglob
eval `tset –s . . .`
unset noglob
```

to bring the terminal information into the environment. Doing so makes programs such as **vi**(1) start up faster. If the **SHELL** environment variable ends with **csh**, C shell commands are output, otherwise Bourne shell commands are output.

–m [*port-ID* [*baudrate*] **:** *type*] . . .

Specify (map) a terminal type when connected to a generic port (such as *dialup* or *plugboard*) identified by *port-ID*. The *baudrate* argument can be used to check the baudrate of the port and set the terminal type accordingly. The target rate is prefixed by any combination of the following operators to specify the conditions under which the mapping is made:

>	Greater than
@	Equals or “at”
<	Less than
!	It is not the case that (negates the above operators)
?	Prompt for the terminal type. If no response is given, then *type* is selected by default.

In the following example, the terminal type is set to **adm3a** if the port is a dialup with a speed of greater than 300 or to **dw2** if the port is a dialup at 300 baud or less. In the third case, the question mark preceding the terminal type indicates that the user is to verify the type desired. A **NULL** response indicates that the named type is correct. Otherwise, the user’s response is taken to be the type desired.

```
tset –m ’dialup>300:adm3a’ –m ’dialup:dw2’ –m \
        ’plugboard:?adm3a’
```

To prevent interpretation as metacharacters, the entire argument to **–m** should be enclosed in single quotes. When using the C shell, exclamation points should be preceded by a backslash (\).

EXAMPLES

These examples all use the ‘–’ option. A typical use of **tset** in a **.profile** or **.login** will also use the –**e** and –**k** options, and often the –**n** or –**Q** options as well. These options have been omitted here to keep the examples short.

To select a 2621, you might put the following sequence of commands in your **.login** file (or **.profile** for Bourne shell users).

```
set noglob
eval `tset –s 2621`
unset noglob
```

If you have a switch which connects to various ports (making it impractical to identify which port you may be connected to), and use various terminals from time to time, you can select from among those terminals according to the *speed* or baud rate. In the example below, **tset** will prompt you for a terminal type if the baud rate is greater than 1200 (say, 9600 for a terminal connected by an RS-232 line), and use a Wyse® 50 by default. If the baud rate is less than or equal to 1200, it will select a 2621. Note the placement of the question mark, and the quotes to protect the **>** and **?** from interpretation by the shell.

```
set noglob
eval `tset –s –m ’switch>1200:?wy’ –m ’switch<=1200:2621’`
unset noglob
```

The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals, and the terminal you use most often is an **adm3a**.

```
set noglob
eval `tset –s ?adm3a`
unset noglob
```

If you want to make the selection based only on the baud rate, you might use the following:

```
set noglob
eval `tset –s –m ’>1200:wy’ 2621`
unset noglob
```

The following example quietly sets the erase character to BACKSPACE, and kill to CTRL-U. If the port is switched, it selects a Concept™ 100 for speeds less than or equal to 1200, and asks for the terminal type otherwise (the default in this case is a Wyse 50). If the port is a direct dialup, it selects Concept 100 as the terminal type. If logging in over the ARPANET, the terminal type selected is a Datamedia® 2500 terminal or emulator. Note the backslash escaping the NEWLINE at the end of the first line in the example.

```
set noglob
eval `tset –e –kˆU –Q –s –m ’switch<=1200:concept100’ –m \
        ’switch:?wy’ –m dialup:concept100 –m arpanet:dm2500`
unset noglob
```

FILES

.login
.profile
/usr/share/lib/terminfo/∗

SEE ALSO

csh(1), **sh**(1), **stty**(1), **vi**(1), **environ**(5)

NOTES

The **tset** command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the **login** program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

This program cannot intuit personal choices for erase, interrupt and line kill characters, so it leaves these set to the local system standards.

It could well be argued that the shell should be responsible for ensuring that the terminal remains in a sane state; this would eliminate the need for the **reset** program.

NAME

tsort – topological sort

SYNOPSIS

/usr/ccs/bin/tsort [*filename*]

DESCRIPTION

The **tsort** command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *filename*. If no *filename* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

tsort returns exit code 1 if the file specified is not found.

NAME

tty – get the name of the terminal

SYNOPSIS

tty [–**l**] [–**s**]

AVAILABILITY

SUNWcsu

DESCRIPTION

tty prints the path name of the user's terminal.

OPTIONS

–l	Prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.
–s	Inhibits printing of the terminal path name, allowing one to test just the exit code.

EXIT CODES

2	if invalid options were specified,
0	if standard input is a terminal,
1	otherwise.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tty** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tty** behaves.

LC_CTYPE

Determines how **tty** handles characters. When **LC_CTYPE** is set to a valid value, **tty** can display and handle text and filenames containing valid characters for that locale. **tty** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tty** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

environ(5)

DIAGNOSTICS

"not on an active synchronous line"

The standard input is not a synchronous terminal and –**l** is specified.

"not a tty" The standard input is not a terminal and –**s** is not specified.

NAME

typeset, type, whence – shell built-in functions to set/get attributes and values for shell variables and functions

SYNOPSIS

sh

type [*name* ...]

ksh

†† **typeset** [±**HLRZfilrtux**[*n*]] [*name*[**=***value*]] ...

whence [–**pv**] *name* ...

DESCRIPTION

sh

For each *name*, the **type** command will indicate how it would be interpreted if used as a command name.

ksh

typeset sets attributes and values for shell variables and functions. When **typeset** is invoked inside a function, a new instance of the variables *name* is created. The variables *value* and *type* are restored when the function completes.
The following list of attributes may be specified:

–**H**	This flag provides UNIX to host-name file mapping on non-UNIX machines.
–**L**	Left justify and remove leading blanks from *value*. If *n* is non-zero it defines the width of the field; otherwise, it is determined by the width of the value of first assignment. When the variable is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the –**Z** flag is also set. The –**R** flag is turned off.
–**R**	Right justify and fill with leading blanks. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left filled with blanks or truncated from the end if the variable is reassigned. The –**L** flag is turned off.
–**Z**	Right justify and fill with leading zeros if the first non-blank character is a digit and the –**L** flag has not been set. If *n* is non-zero it defines the width of the field; otherwise, it is determined by the width of the value of first assignment.
–**f**	The names refer to function names rather than variable names. No assignments can be made and the only other valid flags are –**t**, –**u** and –**x**. The flag –**t** turns on execution tracing for this function. The flag –**u** causes this function to be marked undefined. The **FPATH** variable will be searched to find the function definition when the function is referenced. The flag –**x** allows the function definition to remain in effect across shell procedures invoked by name.
–**i**	Parameter is an integer. This makes arithmetic faster. If *n* is non-zero it defines the output arithmetic base; otherwise, the first assignment determines the output base.
–**l**	All upper-case characters are converted to lower-case. The upper-case flag, –**u** is turned off.
–**r**	The given *name*s are marked **readonly** and these names cannot be changed by subsequent assignment.

–t Tags the variables. Tags are user definable and have no special meaning to the shell.

–u All lower-case characters are converted to upper-case characters. The lower-case flag, **–l** is turned off.

–x The given *name*s are marked for automatic export to the **environment** of subsequently-executed commands.

The –**i** attribute can not be specified along with –**R**, –**L**, –**Z**, or –**f**.

Using **+** rather than – causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of *names* (and optionally the *values*) of the *variables* which have these flags set is printed. (Using **+** rather than – keeps the values from being printed.) If no *name*s and flags are given, the *names* and *attributes* of all *variables* are printed.

For each *name*, **whence** indicates how it would be interpreted if used as a command name.

The –**v** flag produces a more verbose report.

The –**p** flag does a path search for *name* even if name is an alias, a function, or a reserved word.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

ksh(1), **set**(1), **sh**(1)

NAME

ucblinks – adds /dev entries to give SunOS 4.1 compatible names to SunOS 5.x devices

SYNOPSIS

/usr/ucb/ucblinks [**–e** *rulebase*] [**–r** *rootdir*]

AVAILABILITY

SUNWscpu

DESCRIPTION

ucblinks creates symbolic links under the **/dev** directory for devices whose SunOS 5.x names differ from their SunOS 4.1 names. Where possible, these symbolic links point to the device's SunOS 5.x name rather than to the actual **/devices** entry.

ucblinks does not remove unneeded compatibility links; these must be removed by hand.

ucblinks should be called each time the system is reconfiguration-booted, after any new SunOS 5.x links that are needed have been created, since the reconfiguration may have resulted in more compatibility names being needed.

In releases prior to SunOS 5.4, **ucblinks** used a **nawk** rule-base to construct the SunOS 4.1 compatible names. **ucblinks** no longer uses **nawk** for the default operation, although **nawk** rule-bases can still be specifed with the **–e** option. The **nawk** rule-base equivalent to the SunOS 5.4 default operation can be found in **/usr/ucblib/ucblinks.awk**.

OPTIONS

–e *rulebase*	Specify *rulebase* as the file containing **nawk**(1) pattern-action statements.
–r *rootdir*	Specify *rootdir* as the directory under which **dev** and **devices** will be found, rather than the standard root directory **/**.

FILES

/usr/ucblib/ucblinks.awk	sample rule-base for compatibility links

SEE ALSO

devlinks(1M), **disks**(1M), **ports**(1M), **tapes**(1M)

NAME

ul – do underlining

SYNOPSIS

ul [**–i**] [**–t** *terminal*] [*filename*. . .]

AVAILABILITY

SUNWdoc

DESCRIPTION

ul reads the named *filename*s (or the standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable **TERM**. **ul** uses the **/usr/share/lib/terminfo** entry to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, **ul** degenerates to **cat**(1). If the terminal cannot underline, underlining is ignored.

OPTIONS

–t *terminal*
Override the terminal kind specified in the environment. If the terminal cannot underline, underlining is ignored. If the terminal name is not found, no underlining is attempted.

–i Indicate underlining by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an **nroff**(1) output stream on a CRT-terminal.

RETURN VALUES

ul returns exit code 1 if the file specified is not found.

FILES

/usr/share/lib/terminfo/∗

SEE ALSO

cat(1), **man**(1), **nroff**(1)

BUGS

nroff usually generates a series of backspaces and underlines intermixed with the text to indicate underlining. **ul** makes attempt to optimize the backward motion.

NAME

umask – shell built-in function to restrict read / write / execute permissions

SYNOPSIS

sh

umask [*ooo*]

csh

umask [*ooo*]

ksh

umask [*mask*]

DESCRIPTION

sh

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read / write / execute permissions for owner, group, and other, respectively (see **chmod**(1), **chmod**(2), and **umask**(2)). The value of each specified digit is subtracted from the corresponding ‘‘digit’’ specified by the system for the creation of a file (see **creat**(2)). For example, **umask 022** removes write permission for group and other (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell.

umask can be included in the user’s **.profile** (see **profile**(4)) and invoked at login to automatically set the user’s permissions on files or directories created.

csh

See the description above for the Bourne shell (**sh**) **umask** built-in.

ksh

The user file-creation mask is set to *mask*. *mask* can either be an octal number or a symbolic value as described in **chmod**(1). If a symbolic value is given, the new **umask** value is the complement of the result of applying *mask* to the complement of the previous umask value. If *mask* is omitted, the current value of the mask is printed.

SEE ALSO

chmod(1), **csh**(1), **ksh**(1), **sh**(1), **chmod**(2), **creat**(2). **profile**(4)

NAME

uname – print name of current system

SYNOPSIS

uname [**–amnprsv**]
uname [**–S** *system_name*]

AVAILABILITY

SUNWcsu

DESCRIPTION

uname prints information about the current system on the standard output. If no options are specified, **uname** prints the current operating system's name. The options print selected information returned by **uname**(2) and / or **sysinfo**(2).

OPTIONS

–a	Print all information.
–m	Print the machine hardware name.
–n	Print the nodename (the nodename is the name by which the system is known to a communications network).
–p	Print the current host's processor type.
–r	Print the operating system release.
–s	Print the name of the operating system. This is the default.
–v	Print the operating system version.
–S *systemname*	The nodename may be changed by specifying a system name argument. The system name argument is restricted to SYS_NMLN characters. SYS_NMLN is an implementation specific value defined in **<sys/utsname.h>**. Only the super-user is allowed this capability.

ENVIRONMENT

If any of the **LC_∗** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **uname** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_∗** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **uname** behaves.

LC_CTYPE

Determines how **uname** handles characters. When **LC_CTYPE** is set to a valid value, **uname** can display and handle text and filenames containing valid characters for that locale. **uname** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **uname** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

sysinfo(2), **uname**(2), **environ**(5)

NAME

unifdef – resolve and remove ifdef'ed lines from C program source

SYNOPSIS

unifdef [–**clt**] [–**D***name*] [–**U***name*] [–**iD***name*] [–**iU***name*] . . . [*filename*]

DESCRIPTION

unifdef removes **ifdef**ed lines from a file while otherwise leaving the file alone. It is smart enough to deal with the nested **ifdef**s, comments, single and double quotes of C syntax, but it does not do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined with –**D** options, and which you want undefined with –**U** options. Lines within those **ifdef**s will be copied to the output, or removed, as appropriate. Any **ifdef**, **ifndef**, **else**, and **endif** lines associated with *filename* will also be removed.

ifdefs involving symbols you do not specify are untouched and copied out along with their associated **ifdef**, **else**, and **endif** lines.

If an **ifdef***X* occurs nested inside another **ifdef***X*, then the inside **ifdef** is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

unifdef copies its output to the standard output and will take its input from the standard input if no *filename* argument is given.

OPTIONS

–**c**	Complement the normal operation. Lines that would have been removed or blanked are retained, and vice versa.
–**l**	Replace ‘‘lines removed’’ lines with blank lines.
–**t**	Plain text option. **unifdef** refrains from attempting to recognize comments and single and double quotes.
–**D***name*	Lines associated with the defined symbol *name*.
–**U***name*	Lines associated with the undefined symbol *name*.
–**iD***name*	Ignore, but print out, lines associated with the defined symbol *name*. If you use **ifdef**s to delimit non-C lines, such as comments or code which is under construction, then you must tell **unifdef** which symbols are used for that purpose so that it will not try to parse for quotes and comments within them.
–**iU***name*	Ignore, but print out, lines associated with the undefined symbol *name*.

SEE ALSO

diff(1)

DIAGNOSTICS

Premature EOF	Inappropriate **else** or **endif**.

Exit status is 1 if **unifdef** encounters problems, and 0 otherwise.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [[–**u**] [–**d**] [–**c**] [**+**n] [–n]] [*input* [*output*]]

AVAILABILITY

SUNWesu

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see **sort**(1).

OPTIONS

If the –**d** and the –**u** options are used, the last option used on the command line will supersede the other option.

–**u** If the –**u** flag is used, just the lines that are not repeated in the original file are output.

–**d** The –**d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the –**u** and –**d** mode outputs.

–**c** The –**c** option supersedes –**u** and –**d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

EXAMPLES

The following example lists the contents of the **uniq.test** file and outputs a copy of the repeated lines.

```
example% cat uniq.test
This is a test.
This is a test.
TEST.
Computer.
TEST.
TEST.
Software.

example% uniq –d uniq.test
This is a test.
TEST.
example%
```

The next example outputs just those lines that are not repeated in the **uniq.test** file.

```
example% uniq –u uniq.test
TEST.
Computer.
Software.
example%
```

The last example outputs a report with each line preceded by a count of the number of times each line occurred in the file.

```
example% uniq –c uniq.test
  2 This is a test.
  1 TEST.
  1 Computer.
  2 TEST.
  1 Software.
example%
```

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **uniq** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **uniq** behaves.

LC_CTYPE

Determines how **uniq** handles characters. When **LC_CTYPE** is set to a valid value, **uniq** can display and handle text and filenames containing valid characters for that locale. **uniq** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **uniq** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

comm(1), **sort**(1), **environ**(5)

NAME

units – converts quantities expressed in standard scales to other scales

SYNOPSIS

units

AVAILABILITY

SUNWesu

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have:  inch
You want:  cm
           * 2.540000e+00
           / 3.937008e–01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have:  15 lbs force/in2
You want:  atm
           * 1.020689e+00
           / 9.797299e–01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro’s number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (for example, **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/share/lib/unittab
```

FILES

/usr/share/lib/unittab

NAME

unix2dos – convert text file from ISO format to DOS format

SYNOPSIS

unix2dos [**–ascii**] [**–iso**] [**–7**] *originalfile convertedfile*

AVAILABILITY

SUNWesu

DESCRIPTION

unix2dos converts ISO standard characters to the corresponding characters in the DOS extended character set.

This command may be invoked from either DOS or SunOS. However, the filenames must conform to the conventions of the environment in which the command is invoked.

If the original file and the converted file are the same, **unix2dos** will rewrite the original file after converting it.

OPTIONS

–ascii	Adds carriage returns and converts end of file characters in SunOS format text files to conform to DOS requirements.
–iso	This is the default. Converts ISO standard characters to the corresponding character in the DOS extended character set.
–7	Convert 8 bit SunOS characters to 7 bit DOS characters.

DIAGNOSTICS

File *filename* **not found, or no read permission**

The input file you specified does not exist, or you do not have read permission (check with the SunOS command **ls –l**).

Bad output filename *filename***, or no write permission**

The output file you specified is either invalid, or you do not have write permission for that file or the directory that contains it. Check also that the drive or diskette is not write-protected.

Error while writing to temporary file

An error occurred while converting your file, possibly because there is not enough space on the current drive. Check the amount of space on the current drive using the DIR command. Also be certain that the default diskette or drive is write-enabled (not write-protected). Note that when this error occurs, the original file remains intact.

Could not rename tmpfile to *filename***.**
Translated tmpfile name = *filename***.**

The program could not perform the final step in converting your file. Your converted file is stored under the name indicated on the second line of this message.

SEE ALSO

dos2unix(1)

NAME

uptime – show how long the system has been up

SYNOPSIS

uptime

AVAILABILITY

SUNWcsu

DESCRIPTION

The **uptime** command prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a **w**(1) command.

EXAMPLE

Below is an example of the output **uptime** provides:

```
example% uptime
10:47am  up 27 day(s), 50 mins,  1 user,  load average: 0.18, 0.26, 0.20
```

SEE ALSO

w(1), **who**(1), **whodo**(1M)

NOTES

who –b gives the time the system was last booted.

NAME

users – display a compact list of users logged in

SYNOPSIS

/usr/ucb/users [*filename*]

AVAILABILITY

SUNWscpu

DESCRIPTION

users lists the login names of the users currently on the system in a compact, one-line format.

Specifying *filename*, tells **users** where to find its information; by default it checks **/var/adm/utmp**.

Typing *users* is equivalent to typing **who –q**.

EXAMPLES

```
example% users
paul george ringo
example%
```

FILES

/var/adm/utmp

SEE ALSO

who(1)

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [**–c** | **–C**] [**–d** | **–f**] [**–g***grade*] [**–j**] [**–m**] [**–n***user*] [**–r**]
[**–s***file*] [**–x***debug_level*] *source-file destination-file*

uulog [**–s***sys*] [**–f system**] [**–x**] [*–number*] *system*

uuname [**–c**] [**–l**]

AVAILABILITY

SUNWbnuu

DESCRIPTION

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A source file name may be a path name on your machine, or may have the form:

*system-name***!***pathname*

where *system-name* is taken from a list of system names that **uucp** knows about. *source_file* is restricted to no more than one *system-name*. The destination *system-name* may also include a list of system names such as

*system-name***!***system-name***!...!***system-name***!***pathname*

In this case, an attempt is made to send the file, using the specified route, to the destination. Care should be taken to ensure that intermediate nodes in the route are willing to forward information (see **NOTES** below for restrictions).

For C-Shell users, the ‘‘!’’ character must be surrounded by single quotes (’), or preceded by a backslash (\).

The shell metacharacters **?**, ∗ and **[...]** appearing in *path name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by **~***user* where *user* is a login name on the specified system and is replaced by that user’s login directory;

(3) a path name preceded by **~**/*destination* where *destination* is appended to **/var/spool/uucppublic**; (Note: This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a ’/’. For example **~/dan/** as the destination will make the directory **/var/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory).

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

Invoking **uucp** with shell wildcard characters as the remote *source-file* invokes the **uux**(1C) command to execute the **uucp** command on the remote machine. The remote **uucp** command spools the files on the remote machine. After the first session terminates, if the remote machine is configured to transfer the spooled files to the local machine, the remote machine will initiate a call and send the files; otherwise, the user must "call" the remote machine to transfer the files from the spool directory to the local machine. This call can be done manually using **Uutry**(1M), or as a side effect of another **uux**(1C) or **uucp** call.

Note that the local machine has to have permission to execute the uucp command on the remote machine in order for the remote machine to send the spooled files.

uucp removes execute permissions across the transmission and gives **0666** read and write permissions (see **chmod**(2)).
uulog queries a log file of **uucp** or **uuxqt** transactions in file **/var/uucp/.Log/uucico/system or /var/uucp/.Log/uuxqt/***system*.

uuname lists the names of systems known to **uucp**.

OPTIONS

The following options are interpreted by **uucp**:

–c	Do not copy local file to the spool directory for transfer to the remote machine (default).
–C	Force the copy of local files to the spool directory for transfer.
–d	Make all necessary directories for the file copy (default).
–f	Do not make intermediate directories for the file copy.
–g*grade*	*grade* can be either a single letter, number, or a string of alphanumeric characters defining a service grade. The **uuglist** command can determine whether it is appropriate to use the single letter, number, or a string of alphanumeric characters as a service grade. The output from the **uuglist** command will be a list of service grades that are available, or a message that says to use a single letter or number as a grade of service.
–j	Print the **uucp** job identification string on standard output. This job identification can be used by **uustat** to obtain the status of a **uucp** job or to terminate a **uucp** job. The **uucp** job is valid as long as the job remains queued on the local system.
–m	Send mail to the requester when the copy is complete.
–n*user*	Notify *user* on the remote system that a file was sent.
–r	Do not start the file transfer, just queue the job.
–s*file*	Report status of the transfer to *file*. This option is accepted for compatibility, but it is ignored because it is insecure.
–x*debug_level*	Produce debugging output on standard output. *debug_level* is a number between 0 and 9; as it increases to 9, more detailed debugging information is given. This option may not be available on all systems.

The following options cause **uulog** to print logging information:

–s*sys*	Print information about file transfer work involving system *sys*.
–f*system*	Do a "**tail –f**" of the file transfer log for **system**. (You must hit BREAK to

exit this function.)

Other options used in conjunction with the above options are:

−x	Look in the **uuxqt** log file for the given system.
−*number*	Execute a **tail** command of *number* lines.

uuname options are:

−c	Display the names of systems known to **cu**. The two lists are the same, unless your machine is using different **Systems** files for **cu** and **uucp**. See the **Sysfiles** file.
−l	Display the local system name.

FILES

/etc/uucp/∗	other data files
/var/spool/uucp	spool directories
/usr/lib/uucp/∗	other program files
/var/spool/uucppublic/∗	public directory for receiving and sending

SEE ALSO

mail(1), **uuglist**(1C), **uustat**(1C), **uux**(1C), **Uutry**(1M), **uuxqt**(1M), **chmod**(2)

NOTES

For security reasons, the domain of remotely accessible files may be severely restricted. You will probably not be able to access files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/var/spool/uucppublic** (equivalent to **~/**).

All files received by **uucp** will be owned by **uucp**.

The −**m** option will only work when sending files or receiving a single file. Receiving multiple files specified by special shell characters **?**, **&**, and **[. . .]** will not activate the −**m** option.

The forwarding of files through other systems may not be compatible with the previous version of **uucp**. If forwarding is used, all systems in the route must have compatible versions of **uucp**.

Protected files and files that are in protected directories that are owned by the requester can be sent by **uucp**. However, if the requester is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uuencode, uudecode – encode a binary file, or decode its ASCII representation

SYNOPSIS

uuencode [*source-file*] *file-label*

uudecode [**–p**] [*encoded-file*]

AVAILABILITY

SUNWesu

DESCRIPTION

uuencode converts a binary file into an ASCII-encoded representation that can be sent using **mail**(1). It encodes the contents of *source-file*, or the standard input if no *source-file* argument is given. The *file-label* argument is required. The *file-label* is included in the encoded file's header as the name of the file into which **uudecode** is to place the binary (decoded) data. **uuencode** also includes the permission modes of *source-file*, (except **setuid**, **setgid**, and sticky-bits), so that *file-label* is recreated with those same permission modes.

uudecode reads an *encoded-file*, strips off any leading and trailing lines added by mailer programs, and recreates the original binary data with the filename and the mode specified in the header.

The encoded file is an ordinary ASCII text file; it can be edited by any text editor. But it is best only to change the mode or file-label in the header to avoid corrupting the decoded binary.

OPTIONS

–p	decode *encoded-file* and send it to standard output. This allows **uudecode** to be used in a pipeline.

SEE ALSO

mail(1), **uucp**(1C), **uux**(1C)

NOTES

The encoded file's size is expanded by 35% (3 bytes become 4, plus control information), causing it to take longer to transmit than the equivalent binary.

The user on the remote system who is invoking **uudecode** (typically **uucp**) must have write permission on the file specified in the *file-label*.

If you **uuencode** then **uudecode** a file in the same directory, you will overwrite the original file.

NAME

uuglist – print the list of service grades that are available on this UNIX system

SYNOPSIS

uuglist [–**u**]

AVAILABILITY

SUNWbnuu

DESCRIPTION

uuglist prints the list of service grades that are available on the system to use with the –**g** option of **uucp**(1C) and **uux**(1C).

OPTIONS

–**u**	List the names of the service grades that the user is allowed to use with the –**g** option of the **uucp** and **uux** commands.

FILES

/etc/uucp/Grades	contains the list of service grades

SEE ALSO

uucp(1C), **uux**(1C)

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [–**m**] | [–**p**] | [–**q**] | [–**k***jobid* [–**n**]] | [–**r***jobid* [–**n**]]
uustat [–**a**] [–**s***system* [–**j**]] [–**u***user*] [–**S***qric*]
uustat –**t***system* [–**c**] [–**d***number*]

AVAILABILITY

SUNWbnuu

DESCRIPTION

uustat functions in the following three areas: displays the general status of, or cancels, previously specified **uucp** commands; provides remote system performance information, in terms of average transfer rates or average queue times; and provides general remote system-specific and user-specific status of **uucp** connections to other systems.

OPTIONS

Here are the options that obtain general status of, or cancel, previously specified **uucp** commands:

–**a**	List all jobs in queue.
–**j**	List the total number of jobs displayed. The –**j** option can be used in conjunction with the –**a** or the –**s** option.
–**k***jobid*	Kill the **uucp** request whose job identification is *jobid*. The killed **uucp** request must belong to the user issuing the **uustat** command unless the user is the super-user or uucp administrator. If the job is killed by the super-user or uucp administrator, electronic mail is sent to the user.
–**m**	Report the status of accessibility of all machines.
–**n**	Suppress all standard output, but not standard error. The –**n** option is used in conjunction with the –**k** and –**r** options.
–**p**	Execute the command **ps** –**flp** for all the process-ids that are in the lock files.
–**q**	List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in parentheses next to the number of **C** or **X** files, it is the age in days of the oldest **C.**/**X.** file for that system. The **Retry** field represents the number of hours until the next possible call. The **Count** is the number of failure attempts. Note: For systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. An example of the output produced by the –**q** option is:

```
eagle     3C   04/07-11:07     NO DEVICES AVAILABLE
mh3bs3  2C   07/07-10:42     SUCCESSFUL
```

This indicates the number of command files that are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

–**r***jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs' modification time reaches the limit imposed by the daemon.

Here are the options that provide remote system performance information, in terms of average transfer rates or average queue times; the –**c** and –**d** options can only be used in conjunction with the –**t** option:

–**t***system* Report the average transfer rate or average queue time for the past 60 minutes for the remote *system*. The following parameters can only be used with this option:

–**c** Average queue time is calculated when the –**c** parameter is specified and average transfer rate when –**c** is not specified. For example, the command:

```
example% uustat –teagle –d50 –c
```

produces output in the following format:

```
average queue time to eagle for last 50 minutes: 5 seconds
```

The same command without the –**c** parameter produces output in the following format:

```
average transfer rate with eagle for last 50 minutes: 2000.88 bytes/sec
```

–**d***number* *number* is specified in minutes. Used to override the 60 minute default used for calculations. These calculations are based on information contained in the optional performance log and therefore may not be available. Calculations can only be made from the time that the performance log was last cleaned up.

Here are the options that provide general remote system-specific and user-specific status of **uucp** connections to other systems. Either or both of the following options can be specified with **uustat.** The –**j** option can be used in conjunction with the –**s** option to list the total number of jobs displayed:

–**s***system* Report the status of all **uucp** requests for remote system *system.*

–**u***user* Report the status of all **uucp** requests issued by *user*.

Output for both the –**s** and –**u** options has the following format:

```
eagleN1bd7  4/07-11:07   S   eagle   dan   522     /home/dan/A
eagleC1bd8  4/07-11:07   S   eagle   dan   59      D.3b2al2ce4924
            4/07-11:07   S   eagle   dan   rmail   mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date / time. The next field is an **S** if the job is sending a file or an **R** if the job is requesting a file. The next field is the machine where the file is to be transferred. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (**rmail** is the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (for example, **D.3b2alce4924**) that is created for data files associated with remote executions (**rmail** in this example).

–**S***qric* Report the job state: **q** for queued jobs, **r** for running jobs, **i** for interrupted jobs, and **c** for completed jobs.

A job is queued if the transfer has not started. A job is running when the transfer has begun. A job is interrupted if the transfer began but was terminated before the file was completely transferred. A completed job is a job that successfully transferred. The completed state information is maintained in the accounting log, which is optional and therefore may be unavailable. The parameters can be used in any combination, but at least one parameter must be specified. The –**S** option can also be used with –**s** and –**u** options. The output for this option is exactly like the output for –**s** and –**u** except that the job states are appended as the last output word. Output for a completed job has the following format:

eagleC1bd3 completed

When no options are given, **uustat** outputs the status of all **uucp** requests issued by the current user.

FILES

/var/spool/uucp/∗	spool directories
/var/uucp/.Admin/account	accounting log
/var/uucp/.Admin/perflog	performance log

SEE ALSO

uucp(1C)

DIAGNOSTICS

The –**t** option produces no message when the data needed for the calculations is not being recorded.

NOTES

After the user has issued the **uucp** request, if the file to be transferred is moved, deleted or was not copied to the spool directory (–**C** option) when the **uucp** request was made, **uustat** reports a file size of –99999. This job will eventually fail because the file(s) to be transferred can not be found.

NAME

uuto, uupick – public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [*options*] *source-files destination*

uupick [**–s** *system*]

AVAILABILITY

SUNWbnuu

DESCRIPTION

uuto sends *source-files* to *destination*. **uuto** uses the **uucp**(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system[*!system*] ... *!user*

where **system** is taken from a list of system names that **uucp** knows about. *User* is the login name of someone on the specified system.

The files (or sub-trees if directories are specified) are sent to *PUBDIR* on **system**, where *PUBDIR* is a public directory defined in the **uucp** source. By default, this directory is **/var/spool/uucppublic**. Specifically the files are sent to

PUBDIR / receive / *user* / *mysystem* / files.

The recipient is notified by **mail**(1) of the arrival of files.

OPTIONS

The following options apply to **uuto**:

–p	Copy the source file into the spool directory before transmission.
–m	Send mail to the sender when the copy is complete.

The following option applies to **uupick**:

–s *system*	Search only the *PUBDIR* for files sent from **system**.

USAGE

uupick accepts or rejects the files transmitted to the user. Specifically, **uupick** searches *PUBDIR* for files destined for the user. For each entry (file or directory) found, the following message is printed on standard output:

from system *sysname*: [file *file-name*] [dir *dirname*] **?**

uupick then reads a line from standard input to determine the disposition of the file:

<new-line>	Go to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete path name (in which *$HOME* is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as **m** above, except it moves all the files sent from **system**.

p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

FILES

PUBDIR **/var/spool/uucppublic** public directory

SEE ALSO

mail(1), **uucp**(1C), **uustat**(1C), **uux**(1C), **uucleanup**(1M)

NOTES

In order to send files that begin with a dot (for instance, **. profile**), the files must be qualified with a dot. For example, the following files are correct:

.profile **.prof*** **.profil?**

The following files are incorrect:

prof **?profile**

NAME

uux – UNIX-to-UNIX system command execution

SYNOPSIS

uux [–] [–**a***name*] [–**b**] [–**c**] [–**C**] [–**g***grade*] [–**j**] [–**n**] [–**p**] [–**r**] [–**s***filename*] [–**x***debug_level*] [–**z**] *command-string*

AVAILABILITY

SUNWbnuu

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

Note: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from **uux**, permitting only the receipt of mail (see **mail**(1)). (Remote execution permissions are defined in **/etc/uucp/Permissions**.)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of:

- a full pathname;
- a pathname preceded by ˜*xxx*, where *xxx* is a login name on the specified system and is replaced by that user's login directory;
- anything else is prefixed by the current directory.

As an example, the command:

```
example% uux "!diff sys1!/home/dan/filename1 sys2!/a4/dan/filename2 >
!˜/dan/filename.diff"
```

will get the *filename1* and *filename2* files from the ''sys1'' and ''sys2'' machines, execute a **diff**(1) command and put the results in *filename.diff* in the local *PUBDIR* / dan / directory. *PUBDIR* is a public directory defined in the **uucp** source. By default, this directory is **/var/spool/uucppublic**.

Any special shell characters such as **<, >, ;, |** should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all appropriate files to the specified system where they will be processed. For files that are output files, the file name must be escaped using parentheses. For example, the command:

```
example% uux "a!cut -f1 b!/usr/filename > c!/usr/filename"
```

gets "**/usr/filename**" from system "**b**" and sends it to system "**a**", performs a **cut** command on that file and sends the result of the **cut** command to system "**c**".

uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the –**n** option. The response comes by remote mail from the remote machine.

OPTIONS

–	The standard input to **uux** is made the standard input to the *command-string*.
–**a***name*	Use *name* as the user job identification replacing the initiator user-id. (Notification will be returned to user-id *name.)*
–**b**	Return whatever standard input was provided to the **uux** command if the exit status is non-zero.
–**c**	Do not copy local file to the spool directory for transfer to the remote machine (default).
–**C**	Force the copy of local files to the spool directory for transfer.
–**g***grade*	*grade* can be either a single letter, number, or a string of alphanumeric characters defining a service grade. The **uuglist**(1C) command determines whether it is appropriate to use the single letter, number, or a string of alphanumeric characters as a service grade. The output from the *uuglist* command will be a list of service grades that are available or a message that says to use a single letter or number as a grade of service.
–**j**	Output the jobid string on the standard output which is the job identification. This job identification can be used by **uustat**(1C) to obtain the status or terminate a job.
–**n**	Do not notify the user if the command fails.
–**p**	Same as –: The standard input to **uux** is made the standard input to the *command-string*.
–**r**	Do not start the file transfer, just queue the job.
–**s** *filename*	Report status of the transfer in *filename*. This option is accepted for compatibility, but it is ignored because it is insecure.
–**x***debug_level*	Produce debugging output on the standard output. *debug_level* is a number between 0 and 9; as it increases to 9, more detailed debugging information is given.
–**z**	Send success notification to the user.

FILES

/etc/uucp/∗	other data and programs
/etc/uucp/Permissions	remote execution permissions
/usr/lib/uucp/∗	other programs
/var/spool/uucp	spool directories

SEE ALSO

cut(1), **mail**(1), **uucp**(1C), **uuglist**(1C), **uustat**(1C)

NOTES

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter ∗ will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the **uucp** system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the **uux** request. The following command will NOT work:

```
example% uux "a!diff b!/home/dan/xyz c!/home/dan/xyz > !xyz.diff"
```

but the command:

```
example% uux "a!diff a!/home/dan/xyz c!/home/dan/xyz > !xyz.diff"
```

will work. (If **diff** is a permitted command.)

Protected files and files that are in protected directories that are owned by the requester can be sent in commands using **uux**. However, if the requester is root, and the directory is not searchable by "other", the request will fail.

NAME

vacation – reply to mail automatically

SYNOPSIS

vacation [**–I**]
vacation [**–j**] [**–a** *alias*] [**–t***N*] *username*

AVAILABILITY

SUNWcsu

DESCRIPTION

vacation automatically replies to incoming mail.

Installation

The installation consists of an interactive program which sets up **vacation**'s basic configuration.

To install **vacation**, type it with no arguments on the command line. The program creates a **.vacation.msg** file, which contains the message that is automatically sent to all senders when **vacation** is enabled, and starts an editor for you to modify the message. (See **USAGE** section.) Which editor is invoked is determine by the **VISUAL** or **EDITOR** environment variable, or **vi**(1) if neither of those environment variables are set.

A **.forward** file is also created if one does not exist in your home directory. Once created, the **.forward** file will contain a line of the form:

\username, "**|/usr/bin/vacation** *username*"

One copy of an incoming message is sent to the *username* and another copy is piped into **vacation.**

If a **.forward** file is present in your home directory, it will ask whether you want to remove it, which disables **vacation** and ends the installation.

The program automatically creates **.vacation.pag** and **.vacation.dir**, which contain a list of senders when **vacation** is enabled.

Activation and Deactivation

The presence of the **.forward** file determines whether or not **vacation** is disabled or enabled. To disable **vacation** remove the **.forward** file, or move it to a new name.

Initialization

vacation –I clears the **vacation** log files, **.vacation.pag** and **.vacation.dir**, erasing the list of senders from a previous **vacation** session. (See **OPTIONS** section).

Additional Configuration

vacation provides configuration options that are not part of the installation, these being **–j**, **–a**, **–t**. (See **OPTIONS** section).

OPTIONS

–I Initialize the **.vacation.pag** and **.vacation.dir** files and enables **vacation**. If the **–I** flag is not specified, and a *user* argument is given, **vacation** reads the first line from the standard input (for a **From:** line, no colon). If absent, it produces an error message.

Options –**j**, –**a**, –**t** are configuration options to be used in conjunction with **vacation** in the **.forward** file, not on the command line. For example,

```
\username, "|/usr/bin/vacation –t1m username"
```

repeats replies to the sender every minute.

–**j**	Do not check whether the recipient appears in the **To:** or the **Cc:** line.
–**a** *alias*	Indicate that *alias* is one of the valid aliases for the user running **vacation**, so that mail addressed to that alias generates a reply.
–**t***N*	Change the interval between repeat replies to the same sender. The default is 1 week. A trailing **s**, **m**, **h**, **d**, or **w** scales *N* to seconds, minutes, hours, days, or weeks respectively.

USAGE

Files

.vacation.msg should include a header with at least a **Subject:** line (it should not include a **From:** or a **To:** line). For example:

```
Subject: I am on vacation
I am on vacation until July 22.  If you have something urgent,
please contact Joe Jones (jones@fB0).
        --John
```

If the string **$SUBJECT** appears in the **.vacation.msg** file, it is replaced with the subject of the original message when the reply is sent; thus, a **.vacation.msg** file such as

```
Subject: I am on vacation
I am on vacation until July 22.
Your mail regarding "$SUBJECT" will be read when I return.
If you have something urgent, please contact
Joe Jones (jones@fB0).
        --John
```

will include the subject of the message in the reply.

No message is sent if the **To:** or the **Cc:** line does not list the user to whom the original message was sent or one of a number of aliases for them, if the initial **From** line includes the string –**REQUEST@**, or if a **Precedence: bulk** or **Precedence: junk** line is included in the header.

vacation will also not respond to mail from either **postmaster** or **Mailer-Daemon**.

FILES

˜/.forward
˜/.vacation.msg

A list of senders is kept in the **dbm** format files **.vacation.pag** and **.vacation.dir** in your home directory. These files are **dbm** files and cannot be viewed directly with text editors.

SEE ALSO

vi(1), **sendmail**(1M), **dbm**(3B), **aliases**(4)

NAME

vc – version control

SYNOPSIS

vc [–**a**] [–**t**] [–**c***char*] [–**s**] [*keyword=value ... keyword=value*]

DESCRIPTION

This command is obsolete and will be removed in the next release.

The **vc** command copies lines from the standard input to the standard output under control of its arguments and of ‘‘control statements’’ encountered in the standard input. In the process of performing the copy operation, user-declared *keyword*s may be replaced by their string *value* when they appear in plain text and / or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as **vc** command arguments.

A control statement is a single line beginning with a control character, except as modified by the –**t** keyletter (see below). The default control character is colon (**:**), except as modified by the –**c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with **ed**; a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The –**a** keyletter (see below) forces replacement of keywords in all lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

OPTIONS

–**a**	Forces replacement of keywords surrounded by control characters with their assigned value in all text lines and not just in **vc** statements.
–**t**	All characters from the beginning of a line up to and including the first tab character are ignored for the purpose of detecting a control statement. If a control statement is found, all characters up to and including the tab are discarded.
–**c***char*	Specifies a control character to be used in place of the ‘‘**:**’’ default.
–**s**	Silences warning messages (not error) that are normally printed on the diagnostic output.

vc recognizes the following version control statements:

:dcl *keyword*[**,** ...**,** *keyword*]

Declare keywords. All keywords must be declared.

:asg *keyword***=***value*

Assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the **vc** command line and all previous **asg** statements for that keyword. Keywords that are declared but are not assigned values have null values.

:if *condition*

. . .

:end

Skip lines of the standard input. If the condition is true, all lines between the **if** statement and the matching **end** statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note: Intervening **if** statements and matching **end** statements are recognized solely for the purpose of maintaining the proper **if-end** matching.

The syntax of a condition is:

<cond> ::= [‘‘**not**’’] *<or>*
<or> ::= *<and>* | *<and>* ‘‘**|**’’ *<or>*
<and> ::= *<exp>* | *<exp>* ‘‘**&**’’ *<and>*
<exp> ::= ‘‘**(**’’ *<or>* ‘‘**)**’’ | *<value>* *<op>* *<value>*
<op> ::= ‘‘**=**’’ | ‘‘**!=**’’ | ‘‘**<**’’ | ‘‘**>**’’
<value> ::= *<arbitrary ASCII string>* | *<numeric string>*

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the **if**, and when present, inverts the value of the entire condition

The **>** and **<** operate only on unsigned integer values (for example, **: 012 > 12** is false). All other operators take strings as arguments (for example, **: 012 != 12** is true).

The precedence of the operators (from highest to lowest) is:

= != > < all of equal precedence
&
|

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

::*text* Replace keywords on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **−a** keyletter.

:on
:off Turn on or off keyword replacement on all lines.

:ctl *char*
Change the control character to *char*.

:msg *message*
Print *message* on the diagnostic output.

:err *message*
Print *message* followed by**:**

ERROR: err statement on line ... (915)

on the diagnostic output. **vc** halts execution, and returns an exit code of 1.

SEE ALSO

ed(1)

NAME

vgrind – grind nice program listings

SYNOPSIS

vgrind [**–2fntwWx**] [**–d** *defs-file*] [**–h** *header*] [**–l***language*] [**–s***n*] [**–o***pagelist*]
[**–P***printer*] [**–T***output-device*] *filename*. . .

AVAILABILITY

SUNWdoc

DESCRIPTION

vgrind formats the program sources named by the *filename* arguments in a nice style using **troff**(1). Comments are placed in italics, keywords in bold face, and as each function is encountered its name is listed on the page margin.

vgrind runs in two basic modes, filter mode or regular mode. In filter mode **vgrind** acts as a filter in a manner similar to **tbl**(1). The standard input is passed directly to the standard output except for lines bracketed by the **troff-**like macros:

.vS	starts processing
.vE	ends processing

These lines are formatted as described above. The output from this filter can be passed to **troff** for output. There need be no particular ordering with **eqn**(1) or **tbl**.

In regular mode **vgrind** accepts input *filename*s, processes them, and passes them to **troff** for output. If no *filename* is given, or if the '–' argument is given, **vgrind** reads from the standard input (default if **–f** is specified).

In both modes **vgrind** passes any lines beginning with a decimal point without conversion.

OPTIONS

Note: The syntax of options with arguments is important. Some require a SPACE between the option name and the argument, while those that do not have a SPACE below will not tolerate one.

–2	Produce two column output. Specifying this option changes the default point size to 8 (as if the **–s8** option were supplied). It also arranges for output to appear in landscape mode, by supplying the **–L** flag to the formatter and changing the page height and width accordingly.
–f	Force filter mode.
–n	Do not make keywords boldface.
–w	Consider TAB characters to be spaced four columns apart instead of the usual eight.
–x	Output the index file in a "pretty" format. The index file itself is produced whenever **vgrind** is run with a file called **index** present in the current directory. The index of function definitions can then be run off by giving **vgrind** the **–x** option and the file **index** as argument.

–d *defs-file*	Specify an alternate language definitions file (default is **/usr/lib/vgrindefs**).
–h *header*	Specify a header to appear in the center of every output page.
–l*language*	Specify the language to use. Among the languages currently known are: Bourne shell (**–lsh**), C (**–lc**, the default), C++ (**–lc++**), C shell (**–lcsh**), emacs MLisp, (**–lml**), FORTRAN (**–lf**), Icon (**–lI**), ISP (**–i**), LDL (**–lLDL**), Model (**–lm**), Pascal (**–lp**), and RATFOR (**–lr**).
–s*n*	Specify a point size to use on output (exactly the same as the argument of a **troff .ps** point size request).

vgrind passes the following options to the formatter specified by the **TROFF** environment variable, see **ENVIRONMENT** below.

–t	Similar to the same option in **troff**; that is, formatted text goes to the standard output.
–W	Force output to the (wide) Versatec printer rather than the (narrow) Varian.
–o*pagelist*	Print only those pages whose page numbers appear in the comma-separated *pagelist* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end.
–P*printer*	Send output to the named *printer*.
–T*output-device*	Format output for the specified *output-device*.

ENVIRONMENT

In regular mode **vgrind** feeds its intermediate output to the text formatter given by the value of the **TROFF** environment variable, or to **troff** if this variable is not defined in the environment. This mechanism allows for local variations in **troff**'s name.

FILES

index	file where source for index is created
/usr/lib/vgrindefs	language descriptions
/usr/lib/vfontedpr	preprocessor
/usr/share/lib/tmac/tmac.vgrind	macro package

SEE ALSO

troff(1)

BUGS

vgrind assumes that a certain programming style is followed:

C	Function names can be preceded on a line only by SPACE, TAB, or an asterisk. The parenthesized arguments must also be on the same line.
FORTRAN	Function names need to appear on the same line as the keywords *function* or *subroutine*.
MLisp	Function names should not appear on the same line as the preceding *defun*.
Model	Function names need to appear on the same line as the keywords *is beginproc*.

Pascal	Function names need to appear on the same line as the keywords *function* or *procedure*.

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of SPACE characters to align source code fails miserably; if you plan to **vgrind** your program you should use TAB characters. This is somewhat inevitable since the fonts **vgrind** uses are variable width.

The mechanism of **ctags**(1) in recognizing functions should be used here.

The −**w** option is a crock, but there is no other way to achieve the desired effect.

The macros defined in **tmac.vgrind** do not coexist gracefully with those of other macro packages, making filter mode difficult to use effectively.

vgrind does not process certain special characters in **csh**(1) scripts correctly.

The **tmac.vgrind** formatting macros wire in the page height and width used in two column mode, effectively making two column output useless for paper sizes other than the standard American size of 8.5 by 11 inches. For other paper sizes, it is necessary to edit the size values given in **tmac.vgrind**. A better solution would be to create a **troff** output device specification intended specifically for landscape output and record size information there.

NAME

vi, view, vedit – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [– | –**s**] [–**l**] [–**L**] [–**R**] [–**r** [*filename*]] [–**t** *tag*] [–**v**] [–**V**] [–**x**] [–**w***n*] [–**C**]
[+*command* | –**c** *command*] *filename*. . .

view [– | –**s**] [–**l**] [–**L**] [–**R**] [–**r** [*filename*]] [–**t** *tag*] [–**v**] [–**V**] [–**x**] [–**w***n*] [–**C**]
[+*command* | –**c** *command*] *filename*. . .

vedit [– | –**s**] [–**l**] [–**L**] [–**R**] [–**r** [*filename*]] [–**t** *tag*] [–**v**] [–**V**] [–**x**] [–**w***n*] [–**C**]
[+*command* | –**c** *command*] *filename*. . .

AVAILABILITY

SUNWcsu

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor **ex**. It is possible to use the command mode of **ex** from within **vi** and vice-versa. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all **ex** line editor commands are described on the **ex**(1) manual page.

When using **vi**, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

OPTIONS

Invocation Options

The following invocation options are interpreted by **vi** (previously documented options are discussed in the NOTES section of this manual page):

– | –**s**	Suppress all interactive user feedback. This is useful when processing editor scripts.
–**l**	Set up for editing LISP programs.
–**L**	List the name of all files saved as the result of an editor or system crash.
–**R**	**Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.
–**r** *filename*	Edit *filename* after an editor or system crash. (Recovers the version of *filename* that was in the buffer when the crash occurred.)
–**t** *tag*	Edit the file containing the *tag* and position the editor at its definition.
–**v**	Start up in display editing state using **vi**. You can achieve the same effect by simply typing the –**vi** command itself.
–**V**	Verbose. Any non-tty input will be echoed on standard error. This may be useful when processing editor commands within shell scripts.
–**x**	Encryption option; when used, **vi** simulates the **X** command of **ex** and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of the **crypt** command. The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the –**x** option.

–w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
–C	Encryption option; same as the **–x** option, except that **vi** simulates the **C** command of **ex**. The **C** command is like the **X** command of **ex**, except that all text read in is assumed to have been encrypted.
+*command* | **–c** *command*	Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *filename* argument indicates one or more files to be edited.

The *view* invocation is the same as **vi** except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. It is the same as **vi** except that the **report** flag is set to 1, the **showmode** and **novice** flags are set, and **magic** is turned off. These defaults make it easier to learn how to use **vi**.

vi Modes

Command	Normal and initial mode. Other modes return to command mode upon completion. **ESC** (escape) is used to cancel a partial command.
Input	Entered by setting any of the following options: **a A i I o O c C s S R**. Arbitrary text may then be entered. Input mode is normally terminated with **ESC** character, or, abnormally, with an interrupt.
Last line	Reading input for **: / ?** or **!**; terminate by typing a carriage return; an interrupt cancels termination.

COMMAND SUMMARY

In the descriptions, **CR** stands for carriage return and **ESC** stands for the escape key.

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text***ESC**	insert *text*
cw*new***ESC**	change word to *new*
ea*s***ESC**	pluralize word (end of word; append **s**; escape from input state)
x	delete a character
dw	delete a word
dd	delete a line
3dd	delete 3 lines
u	undo previous change
ZZ	exit **vi**, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*cmd***CR**	any **ex** or **ed** command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**

scroll amount	**ˆD ˆU**
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
DEL	(delete or rubout) interrupts

File manipulation

ZZ	if file modified, write and exit; otherwise, exit
:wCR	write back changes
:w ! CR	forced write, if permission originally not valid
:qCR	quit
:q ! CR	quit, discard changes
:e _name_**CR**	edit file _name_
:e ! CR	reedit, discard changes
:e + _name_**CR**	edit, starting at end
:e +_n_**CR**	edit starting at line _n_
:e #CR	edit alternate file
:e ! #CR	edit alternate file, discard changes
:w _name_**CR**	write file _name_
:w ! _name_**CR**	overwrite file _name_
:shCR	run shell, then return
: ! _cmd_**CR**	run _cmd_, then return
:nCR	edit next file in arglist
:n _args_**CR**	specify new arglist
ˆG	show current file and line
:ta _tag_**CR**	position cursor to _tag_

In general, any **ex** or **ed** command (such as _substitute_ or _global_) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

ˆF	forward screen
ˆB	backward screen
ˆD	scroll down half screen
ˆU	scroll up half screen
_n_**G**	go to the beginning of the specified line (end default), where _n_ is a line number
/_pat_	next line matching _pat_
?_pat_	previous line matching _pat_
n	repeat last **/** or **?** command
N	reverse last **/** or **?** command
/_pat_/+_n_	nth line after _pat_
?_pat_**?**−_n_	nth line before _pat_
]]	next section / function
[[	previous section / function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph

}	end of paragraph
%	find matching **( ) {** or **}**

Adjusting the screen

ˆL	clear and redraw window
ˆR	clear and redraw window if **ˆL** is → key
zCR	redraw screen with current line at top of window
z–CR	redraw screen with current line at bottom of window
z .CR	redraw screen with current line at center of window
/*pat*/**z–CR**	move *pat* line to bottom of window
z*n* **.CR**	use *n*-line window
ˆE	scroll window down 1 line
ˆY	scroll window up 1 line

Marking and returning

``	move cursor to previous context
´´	move cursor to first non-white space in line
m*x*	mark current position with the ASCII lower-case letter *x*
`*x*	move cursor to mark *x*
´*x*	move cursor to first non-white space in line marked by *x*

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
–	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

ˆ	first non white-space character
0	beginning of line
$	end of line
l or →	forward
h or ←	backward
ˆH	same as ← (backspace)
space	same as → (space bar)
f*x*	find next *x*
F*x*	find previous **x**
t*x*	move to character prior to next *x*
T*x*	move to character following previous *x*
;	repeat last **f**, **F**, **t**, or **T**
,	repeat inverse of last **f**, **F**, **t**, or **T**
n|	move to column *n*
%	find matching **({)** or **}**

Words, sentences, paragraphs

w	forward a word
b	back a word
e	end of word
)	to next sentence
}	to next paragraph
(	back a sentence
{	back a paragraph
W	forward a blank-delimited word
B	back a blank-delimited word
E	end of a blank-delimited word

Corrections during insert

^H	erase last character (backspace)
^W	erase last word
erase	your erase character, same as **^H** (backspace)
kill	your kill character, erase this line of input
\	quotes your erase and kill characters
ESC	ends insertion, back to command mode
CTRL-C	interrupt, suspends insert mode
^D	backtab one character; reset left margin of *autoindent*
^^D	caret (^) followed by control-d (^D); backtab to beginning of line; do not reset left margin of *autoindent*
0^D	backtab to beginning of line; reset left margin of *autoindent*
^V	quote non-printable character

Insert and replace

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, for example, **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	delete characters before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked; however, if a buffer is named (using the ASCII lower-case letters **a - z**), the text in that buffer is put instead.

3yy	yank 3 lines
3yl	yank 3 characters
p	put back text after cursor
P	put back text before cursor
"*x***p**	put from buffer *x*
"*x***y**	yank to buffer *x*
"*x***d**	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d* **p**	retrieve *d*'th last delete

AUTHOR

vi and **ex** were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **vi** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **vi** behaves.

LC_CTYPE

Determines how **vi** handles characters. When **LC_CTYPE** is set to a valid value, **vi** can display and handle text and filenames containing valid characters for that locale. **vi** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **vi** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_TIME

Determines how **vi** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/tmp	default directory where temporary work files are placed; it can be changed using the **directory** option (see the **ex**(1) **set** command)
/usr/share/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.COREterm/?/*	subset of compiled terminal description database

SEE ALSO

intro(1), **ed**(1), **edit**(1), **ex**(1), **environ**(5)

Solaris Advanced User's Guide

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard (see **intro**(1)). A –**r** option that is not followed with an option-argument has been replaced by –**L** and **+***command* has been replaced by –**c** *command*.

The message **file too large to recover with** –**r option ,** which is seen when a file is loaded, indicates that the file can be edited and saved successfully, but if the editing session is lost, recovery of the file with the –**r** option will not be possible.

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

The editing environment defaults to certain configuration options. When an editing session is initiated, **vi** attempts to read the **EXINIT** environment variable. If it exists, the editor uses the values defined in **EXINIT**, otherwise the values set in **$HOME/.exrc** are used. If **$HOME/.exrc** does not exist, the default values are used.

To use a copy of **.exrc** located in the current directory other than **$HOME**, set the *exrc* option in **EXINIT** or **$HOME/.exrc**. Options set in **EXINIT** can be turned off in a local **.exrc** only if *exrc* is set in **EXINIT** or **$HOME/.exrc**.

Tampering with entries in **/usr/share/lib/terminfo/?/*** or **/usr/share/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs such as **vi** that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

Software tabs using **ˆT** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

NAME

vipw – edit the password file

SYNOPSIS

/usr/ucb/vipw

AVAILABILITY

SUNWscpu

DESCRIPTION

vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The **vi**(1) editor will be used unless the environment variable **VISUAL** or **EDITOR** indicates an alternate editor.

vipw performs a number of consistency checks on the password entry for root, and will not allow a password file with a “mangled” root entry to be installed. It also checks the **/etc/shells** file to verify the login shell for root.

FILES

/etc/ptmp
/etc/shells

SEE ALSO

passwd(1), **vi**(1), **passwd**(4)

NAME

volcancel – cancel user's request for removable media that is not currently in drive

SYNOPSIS

/usr/lib/vold/volcancel [**–n**] [*volume*]

DESCRIPTION

volcancel cancels a user's request to access a particular floppy or CD-ROM file system. This command is useful when the removable media containing the file system is not currently in the drive.

Use the path **/vol/rdsk/***name_of_volume* to specify the volume. If called without a volume name to cancel, **volcancel** checks for Volume Management running.

OPTIONS

–n Display the nickname to the device name translation table.

EXAMPLES

To cancel a request to access an unnamed CD-ROM, use

```
example% /usr/lib/vold/volcancel vol/rdsk/unnamed_cdrom
```

To check if volume management is running, use:

```
example% /usr/lib/vold/volcancel || echo volmgmt not running
```

SEE ALSO

rmmount(1M), **volcheck**(1), **vold**(1M), **volmissing**(1), **rmmount.conf**(4), **vold.conf**(4), **volfs**(7)

NAME

volcheck – check for media in a drive. Default checks all floppy media.

SYNOPSIS

volcheck [**–v**] [**–i** *secs*] [**–t** *secs*] *pathname*

DESCRIPTION

volcheck tells Volume Management to look at each *pathname* in sequence and determine if new media has been inserted in the drive.

The default action is to **volcheck** all floppy drives pointed to by volume management.

OPTIONS

–v Verbose.

–t *secs* Check the named device(s) for the next *secs* seconds. The maximum number of seconds allowed is 28800, which is 8 hours. The frequency of checking is specified by -i. There is no default total time.

–i *secs* Set the frequency of device checking to *secs* seconds. The default is 2 seconds. The minimum frequency is 1 second.

EXAMPLES

```
example% volcheck -v /dev/diskette
/dev/diskette has media
```

asks Volume Management to examine the floppy drive for new media.

```
example% volcheck -i 2 -t 600 /dev/diskette1 &
```

asks Volume Management if there is a floppy in the floppy drive every 2 seconds for 600 seconds (10 minutes).

FILES

/dev/volctl	Volume Management control port

SEE ALSO

eject(1), **volcancel**(1), **volmissing**(1) **rmmount**(1M), **vold**(1M), **rmmount.conf**(4), **vold.conf**(4), **volfs**(7)

WARNINGS

Due to a hardware limitation in many floppy drives, the act of checking for media causes mechanical action in the floppy drive. **Continuous polling of the floppy drive will cause the drive to wear out.** It is recommended that polling the drive only be performed during periods of high use.

NAME

volmissing – notify user that volume requested is not in the CD-ROM or floppy drive

SYNOPSIS

/usr/lib/vold/volmissing [**–c**] [**–p**] [**–s**] [**–m** *alias*]

DESCRIPTION

volmissing informs a user when a requested volume is not available. Depending on the option selected, users are notified through their console window, **syslogd**(1M), or a mail message.

volmissing –p is the default action taken by **vold**(1M), the Volume Management daemon, when it needs to notify a user that the requested volume is not available. If you want to change this default event, modify the **/etc/vold.conf** file. See **vold.conf**(4).

You can change the notification method for your system by editing the **vold.conf** configuration file and providing a new option for **volmissing** in the notify entry under the Events category.

OPTIONS

–c	Send a message to the user's console requesting the volume be inserted. To end the notification without inserting the requested volume, use **volcancel**(1).
–p	All **volmissing** events will be handled through a GUI, provided a window system is running on the console. If this option is specified, and no window system is running, all messages go to the system console.
–s	Send one message to the **syslogd**(1M).
–m *alias*	Send a mail message to the specified mail alias about the missing volume.

FILES

/etc/vold.conf	Volume Management daemon configuration file. Directs the Volume Management daemon to control certain devices, and causes action to be taken when specific criteria is met.
/usr/lib/vold/volmissing_popup	Pop-up used when the –**p** option is supplied and a window system is running.

SEE ALSO

volcancel(1), **volcheck**(1), **rmmount**(1M), **syslogd**(1M), **vold**(1M), **rmmount.conf**(4), **vold.conf**(4), **volfs**(7)

NAME

vsig – synchronize a co-process with the controlling FMLI application

SYNOPSIS

vsig

AVAILABILITY

SUNWesu

DESCRIPTION

The **vsig** executable sends a SIGUSR2 signal to the controlling FMLI process. This signal / alarm causes FMLI to execute the FMLI built-in command **checkworld** which causes all posted objects with a **reread** descriptor evaluating to TRUE to be reread. **vsig** takes no arguments.

EXAMPLES

The following is a segment of a shell program:

```
echo "Sending this string to an FMLI process"
vsig
```

The **vsig** executable will flush the output buffer *before* it sends the SIGUSR2 signal to make sure the string is actually in the pipe created by the **cocreate** function.

SEE ALSO

coproc(1F), **kill**(1), **kill**(2), **signal**(3C)

NOTES

Because **vsig** synchronize with FMLI, it should be used rather than **kill** to send a SIGUSR2 signal to FMLI.

NAME

w – who is logged in, and what are they doing

SYNOPSIS

w [–**hlsuw**] [*user*]

AVAILABILITY

SUNWcsu

DESCRIPTION

The **w** command displays a summary of the current activity on the system, including what each user is doing. The heading line shows the current time, the length of time the system has been up, the number of users logged into the system and the average number of jobs in the run queue over the last 1, 5 and 15 minutes.

The fields displayed are: the users login name, the name of the tty the user is on, the time of day the user logged on (in *hours:minutes*), the idle time—that is, the number of minutes since the user last typed anything (in *hours:minutes*), the CPU time used by all processes and their children on that terminal (in *minutes:seconds*), the CPU time used by the currently active processes (in *minutes:seconds*), the name and arguments of the current process.

If a *user* name is included, output is restricted to that user.

OPTIONS

–h	Suppress the heading.
–l	Produce a long form of output, which is the default.
–s	Produce a short form of output. In the short form, the tty is abbreviated, the login time and CPU times are left off, as are the arguments to commands.
–u	Produces the heading line which shows the current time, the length of time the system has been up, the number of users logged into the system, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes.
–w	Produces a long form of output, which is also the same as the default.

EXAMPLE

```
example% w
10:54am  up 27 day(s), 57 mins,  1 user,  load average: 0.28, 0.26, 0.22
User     tty           login@   idle   JCPU    PCPU   what
ralph    console       7:10am      1   10:05     4:31   w
```

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tar** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tar** behaves.

LC_CTYPE

Determines how **tar** handles characters. When **LC_CTYPE** is set to a valid value, **tar** can display and handle text and filenames containing valid characters for that locale. **tar** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tar** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME
Determines how **tar** handles date and time formats. In the "C" locale, date and time handling follow the U.S. rules.

FILES

/var/adm/utmp

SEE ALSO

ps(1), **who**(1), **whodo**(1M), **utmp**(4)

NOTES

The notion of the ‘‘current process’’ is muddy. The current algorithm is ‘the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal’. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. In cases where no process can be found, **w** prints –.

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is ‘‘charged’’ with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

w does not know about the conventions for detecting background jobs. It will sometimes find a background job instead of the right one.

NAME

wait – shell built-in function to wait for other jobs or processes

SYNOPSIS

sh

wait [*n*]
wait [**%***jobid* . . .]

csh

wait [*n*]

ksh

† **wait** [*job*]

DESCRIPTION

sh

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero. **wait** accepts a job identifier, when Job Control is enabled, and the argument, *jobid*, is preceded by a percent-sign.

The shell itself executes **wait**, without creating a new process. If you get the error message **cannot fork, too many processes**, try using the **wait** command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

csh

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes **wait**, without creating a new process. If you get the error message **cannot fork, too many processes**, try using the **wait** command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

ksh

Wait for the specified *job* and report its termination status. If *job* is not given then all currently active child processes are waited for. The exit status from this command is that of the process waited for.

On this man page, **ksh**(1) commands that are preceded by one or two † (daggers) are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the **=** sign and word splitting and file name generation are not performed.

SEE ALSO

csh(1), **jobs**(1), **ksh**(1), **sh**(1)

NAME

wc – display a count of lines, words and characters in a file

SYNOPSIS

wc [**–cClw**] [*name* ...]

AVAILABILITY

SUNWcsu

DESCRIPTION

wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a string of characters delimited by a SPACE , TAB , or by any other character in the library function **iswspace()** (see **iswalpha**(3I)).

OPTIONS

When *name* is specified on the command line, the names are printed along with the counts.

If no option is specified the default is **–lwc** (count lines, words, and bytes.)

–c Count bytes.

–C Count characters.

–l Count lines.

–w Count words delimited by white space characters or new line characters. Delimiting characters are Extended Unix Code (EUC) characters from any code set defined by **iswspace()**.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **wc** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **wc** behaves.

LC_CTYPE

Determines how **wc** handles characters. When **LC_CTYPE** is set to a valid value, **wc** can display and handle text and filenames containing valid characters for that locale. **wc** can display and handle EUC characters where any individual character can be 1, 2, or 3 bytes wide. **wc** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES

Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

SEE ALSO

setlocale(3C), **iswalpha**(3I), **environ**(5)

NAME

what – extract SCCS version information from a file

SYNOPSIS

what [**–s**] *filename* . . .

DESCRIPTION

what searches each *filename* for occurrences of the pattern **@(#)** that the SCCS **get** command (see **sccs-get**(1)) substitutes for the **%Z%** ID keyword, and prints what follows up to a ", **>**, NEWLINE, ****, or null character.

OPTIONS

–s Stop after the first occurrence of the pattern.

EXAMPLES

For example, if a C program in file **program.c** contains

```
char sccsid[ ] = "@(#)identification information ";
```

and **program.c** is compiled to yield **program.o** and **a.out**, the command:

```
example% what program.c program.o a.out
```

produces:

```
program.c:
        identification information
program.o:
        identification information
a.out:  identification information
```

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **sccsfile**(4)

Programming Utilities Guide

DIAGNOSTICS

Use the SCCS **help** command for explanations (see **sccs-help**(1)).

BUGS

There is a remote possibility that a spurious occurrence of the '**@(#)**' pattern could be found by **what**.

NAME

whatis – display a one-line summary about a keyword

SYNOPSIS

whatis *command* . . .

AVAILABILITY

SUNWdoc

DESCRIPTION

whatis looks up a given *command* and displays the header line from the manual section. You can then run the **man**(1) command to get more information. If the line starts '**name(***section***)** . . .' you can do '**man –s** *section* **name**' to get the documentation for it. Try '**whatis ed**' and then you should do '**man –s 1 ed**' to get the manual page for **ed**(1).

whatis is actually just the **–f** option to the **man**(1) command.

whatis uses the **/usr/share/man/windex** database. This database is created by **catman**(1M). If this database does not exist, **whatis** will fail.

FILES

/usr/share/man/windex	table of contents and keyword database

SEE ALSO

apropos(1), **man**(1), **catman**(1M)

NAME

whereis – locate the binary, source, and manual page files for a command

SYNOPSIS

/usr/ucb/whereis [**–bmsu**] [**–BMS** *directory*. . . **–f**] *filename* . . .

AVAILABILITY

SUNWscpu

DESCRIPTION

whereis locates source / binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form *.ext,* for example, **.c**. Prefixes of **s.** resulting from use of source code control are also dealt with. **whereis** then attempts to locate the desired program in a list of standard places:

/usr/bin
/usr/bin
/usr/5bin
/usr/games
/usr/hosts
/usr/include
/usr/local
/usr/etc
/usr/lib
/usr/share/man
/usr/src
/usr/ucb

OPTIONS

–b	Search only for binaries.
–m	Search only for manual sections.
–s	Search only for sources.
–u	Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus '**whereis –m –u** *' asks for those files in the current directory which have no documentation.
–B	Change or otherwise limit the places where **whereis** searches for binaries.
–M	Change or otherwise limit the places where **whereis** searches for manual sections.
–S	Change or otherwise limit the places where **whereis** searches for sources.
–f	Terminate the last directory list and signals the start of file names, and *must* be used when any of the **–B**, **–M**, or **–S** options are used.

EXAMPLES

Find all files in **/usr/bin** which are not documented in **/usr/share/man/man1** with source in **/usr/src/cmd**:

```
example% cd /usr/ucb
example% whereis –u –M /usr/share/man/man1 –S /usr/src/cmd –f ∗
```

FILES

/usr/src/∗
/usr/{doc,man}/∗
/etc, /usr/{lib,bin,ucb,old,new,local}

SEE ALSO

chdir(2)

BUGS

Since **whereis** uses **chdir**(2) to run faster, pathnames given with the –**M**, –**S**, or –**B** must be full; that is, they must begin with a '/'.

NAME

which – locate a command; display its pathname or alias

SYNOPSIS

which [*filename*] . . .

AVAILABILITY

SUNWcsu

DESCRIPTION

which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's **.cshrc** file.

FILES

~/.cshrc	source of aliases and path values

SEE ALSO

csh(1)

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

BUGS

Only aliases and paths from **~/.cshrc** are used; importing from the current environment is not attempted. Must be executed by **csh**(1), since only **csh** knows about aliases.

To compensate for **~/.cshrc** files in which aliases depend upon the **prompt** variable being set, **which** sets this variable. If the **~/.cshrc** produces output or prompts for input when **prompt** is set, **which** may produce some strange results.

NAME

while, until – shell built-in functions to repetitively execute a set of actions while/until conditions are evaluated TRUE

SYNOPSIS

sh

while *conditions* ; **do** *actions* ; **done**
until *conditions* ; **do** *actions* ; **done**

csh

while (*expr***)**
. . .
end

ksh

while *conditions* ; **do** *actions* ; **done**
until *conditions* ; **do** *actions* ; **done**

DESCRIPTION

sh

A **while** command repeatedly executes the **while** *conditions* and, if the exit status of the last command in the *conditions* list is zero, executes the **do** *actions*; otherwise the loop terminates. If no commands in the **do** *actions* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

csh

While *expr* is true (evaluates to nonzero), repeat commands between the **while** and the matching **end** statement. **break** and **continue** may be used to terminate or continue the loop prematurely. The **while** and **end** must appear alone on their input lines. If the shell's input is a terminal, it prompts for commands with a question-mark until the **end** command is entered and then performs the commands in the loop.

ksh

A **while** command repeatedly executes the **while** *conditions* and, if the exit status of the last command in the *conditions* list is zero, executes the **do** *actions*; otherwise the loop terminates. If no commands in the **do** *actions* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

SEE ALSO

break(1), **csh**(1), **ksh**(1), **sh**(1)

NOTES

Both the Bourne shell, **sh**, and the Korn shell, **ksh**, can use the semicolon and the carriage return interchangeably in their syntax of the **if**, **for**, and **while** built-in commands.

NAME

who – who is on the system

SYNOPSIS

who [**–uTlHqpdbrtas**] [*filename*]

who –qn *x* [*filename*]

who am i

who am I

AVAILABILITY

SUNWcsu

DESCRIPTION

who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the **/var/adm/utmp** file to obtain its information. If *filename* is given, that file (which must be in **utmp**(4) format) is examined. Usually, *filename* will be **/var/adm/wtmp**, which contains a history of all the logins since the file was last created.

who with the **am i** or **am I** option identifies the invoking user.

The general format for output is:

```
name [state] line time [idle] [pid] [comment] [exit]
```

The *name*, *line*, and *time* information is produced by all options except **–q**; the *state* information is produced only by **–T**; the *idle* and *pid* information is produced only by **–u** and **–l**; and the *comment* and **exit** information is produced only by **–a**. The information produced for **–p**, **–d**, and **–r** is explained during the discussion of each option, below.

OPTIONS

With options, **who** can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the **init** process. These options are:

–u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev .** The *time* is the time that the user logged in. The *idle* column contains the number of hours and minutes since activity last occurred on that particular line. A dot (**.**) indicates that the terminal has seen activity in the last minute and is therefore ''current''. If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/sbin/inittab** (see **inittab**(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, and so forth.

–T	This option is the same as the **–s** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A ‘ **+** ’ appears if the terminal is writable by anyone; a ‘ **-** ’ appears if it is not. **root** can write to all lines having a ‘ **+** ’ or a ‘ **-** ’ in the *state* field. If a bad line is encountered, a ‘ **?** ’ is printed.
–l	This option lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field does not exist.
–H	This option will print column headings above the regular output.
–q	This is a quick **who ,** displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.
–p	This option lists any other process which is currently active and has been previously spawned by **init .** The *name* field is the name of the program executed by **init** as found in **/sbin/inittab**. The *state*, **line**, and *idle* fields have no meaning. The *comment* field shows the **id** field of the line from **/sbin/inittab** that spawned this process. See **inittab**(4).
–d	This option displays all processes that have expired and not been respawned by **init .** The **exit** field appears for dead processes and contains the termination and exit values (as returned by **wait**(3B)), of the dead process. This can be useful in determining why a process terminated.
–b	This option indicates the time and date of the last reboot.
–r	This option indicates the current *run-level* of the **init** process. In addition, it produces the process termination status, process id, and process exit status (see **utmp**(4)) under the *idle*, *pid*, and *comment* headings, respectively.
–t	This option indicates the last change to the system clock (via the **date** command) by **root .** See **su**(1M) and **date**(1).
–a	This option processes **/var/adm/utmp** or the named *filename* with all options turned on.
–s	This option is the default and lists only the *name*, *line*, and *time* fields.
–n*x*	This option takes a numeric argument, *x*, which specifies the number of users to display per line. *x* must be at least **1**. The **–n** option must be used with **–q**.

Note to the super-user: after a shutdown to the single-user state, **who** returns a prompt; the reason is that since **/var/adm/utmp** is updated at login time and there is no login in single-user state, **who** cannot report accurately on this state. **who am i ,** however, returns the correct information.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **who** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **who** behaves.

LC_CTYPE

Determines how **who** handles characters. When **LC_CTYPE** is set to a valid value, **who** can display and handle text and filenames containing valid characters for that locale. **who** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **who** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_TIME

Determines how **who** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/sbin/inittab
/var/adm/utmp
/var/adm/wtmp

SEE ALSO

date(1), **login**(1), **mesg**(1), **init**(1M), **su**(1M), **wait**(3B), **inittab**(4), **utmp**(4), **environ**(5)

NAME

whoami – display the effective current username

SYNOPSIS

/usr/ucb/whoami

AVAILABILITY

SUNWscpu

DESCRIPTION

whoami displays the login name corresponding to the current effective user ID. If you have used **su** to temporarily adopt another user, **whoami** will report the login name associated with that user ID. **whoami** gets its information from the **geteuid** and **getpwuid** library routines (see **getuid** and **getpwnam**(3C), respectively).

FILES

/etc/passwd	username data base

SEE ALSO

su(1M), **who**(1), **getuid**(2), **getpwnam**(3C)

NAME

whois – Internet user name directory service

SYNOPSIS

whois [**–h** *host*] *identifier*

AVAILABILITY

SUNWcsu

DESCRIPTION

whois searches for an Internet directory entry for an *identifier* which is either a name (such as ‘‘Smith’’) or a handle (such as ‘‘SRI-NIC’’). To force a name-only search, precede the name with a period; to force a handle-only search, precede the handle with an exclamation point.

To search for a group or organization entry, precede the argument with ∗ (an asterisk). The entire membership list of the group will be displayed with the record.

You may of course use an exclamation point and asterisk, or a period and asterisk together.

EXAMPLES

The command:

```
example% whois Smith
```

looks for the name or handle SMITH.

The command:

```
example% whois !SRI-NIC
```

looks for the handle SRI-NIC only.

The command:

```
example% whois .Smith, John
```

looks for the name JOHN SMITH only.

Adding **. . .** to the name or handle argument will match anything from that point; that is, **ZU . . .** will match ZUL, ZUM, and so on.

NAME

write – write to another user

SYNOPSIS

write *user* [*line*]

AVAILABILITY

SUNWcsu

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message:

> **Message from** *yourname* **(tty**??**)** [*date*]**. . .**

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point **write** writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the **line** argument may be used to indicate which line or terminal to send to (for example, **term/12**); otherwise, the first writable instance of the user found in **/var/adm/utmp** is assumed and the following message posted:

> **user is logged on more than one place.**
> **You are connected to** "*terminal*"**.**
> **Other locations are:**
> *terminal*

Permission to write may be denied or granted by use of the **mesg** command. Writing to others is normally allowed by default. Certain commands, such as the **pr** command, disallow messages in order to prevent interference with their output. However, if the user has super-user privilege, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, **write** calls the shell to execute the rest of the line as a command.

write runs **setgid( )** [see **setuid**(2)] to the group ID **tty**, in order to have write permissions on other user's terminals.

write will detect non-printable characters before sending them to the user's terminal. Control characters will appear as a '**ˆ**' followed by the appropriate ASCII character; characters with the high-order bit set will appear in "meta" notation. For example, '**\003**' is displayed as '**ˆC**' and '**\372**' as '**M−z**'.

The following protocol is suggested for using **write**: when you first **write** to another user, wait for them to **write** back before starting to send. Each person should end a message with a distinctive signal (that is, **(o)** for ''over'') so that the other person knows when to reply. The signal **(oo)** (for ''over and out'') is suggested when conversation is to be terminated.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **write** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **write** behaves.

LC_CTYPE
Determines how **write** handles characters. When **LC_CTYPE** is set to a valid value, **write** can display and handle text and filenames containing valid characters for that locale. **write** can display and handle Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **write** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
Determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S. English).

LC_TIME
Determines how **write** handles date and time formats. In the "C" locale, date and time handling follows the U.S. rules.

FILES

/var/adm/utmp	to find user
/usr/bin/sh	to execute **!**

SEE ALSO

mail(1), **mesg**(1), **pr**(1), **sh**(1), **who**(1), **setuid**(2), **environ**(5)

DIAGNOSTICS

user is not logged on	The person you are trying to **write** to is not logged on.
Permission denied	The person you are trying to **write** to denies that permission (with **mesg**).
Warning: cannot respond, set mesg -y	Your terminal is set to **mesg n** and the recipient cannot respond to you.
Can no longer write to user	The recipient has denied permission (**mesg n**) after you had started writing.

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [*flags*] [*command* [*initial-arguments*]]

AVAILABILITY

SUNWcsu

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/usr/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) escapes the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **–i** flag). Flags **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (for example, **–l** vs. **–n**), the last flag has precedence.

OPTIONS

–**l***number*	*command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank / tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option –**x** is forced.
–**i***replstr*	Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option –**x** is also forced. **{}** is assumed for *replstr* if not specified.

–**n***number*	Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments are used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option –**x** is also coded, each *number* arguments must fit in the *size* limitation, else **xargs** terminates execution.
–**t**	Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
–**p**	Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (–**t**) is turned on to print the command instance to be executed, followed by a **?. . .** prompt. A reply of **y** (optionally followed by anything) executes the command; anything else, including just a carriage return, skips that particular invocation of *command*.
–**x**	**xargs** will terminate if any argument list would be greater than *size* characters; –**x** is forced by the options –**i** and –**l**. When neither of the options –**i**, –**l**, or –**n** are coded, the total length of all arguments must be within the *size* limit.
–**s***size*	The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If –**s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
–**e***eofstr*	*eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if –**e** is not coded. The value –**e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). **xargs** reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs terminates if either it receives a return code of –**1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly **exit** (see **sh**(1)) with an appropriate value to avoid accidentally returning with –**1**.

EXAMPLES

The following examples moves all files from directory $1 to directory $2, and echo each move command just before doing it:

```
example% ls $1 | xargs –i –t mv $1/{} $2/{}
```

The following example combines the output of the parenthesized commands onto one line, which is then echoed to the end of file **log**:

```
example% (logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into **arch** (1.) one at a time, or (2.) many at a time.

```
1. ls | xargs –p –l ar r arch
2. ls | xargs –p –l | xargs ar r arch
```

The following example executes **diff**(1) with successive pairs of arguments originally typed as shell arguments:

```
example% echo $* | xargs −n2 diff
```

SEE ALSO

sh(1)

NAME

xgettext – extract gettext call strings from C programs

SYNOPSIS

xgettext [**–ns**] [**–a** [**–x** *exclude-file*]] [**–c** *comment-tag*] [**–d** *default-domain*] [**–j**]
[**–m** *prefix*] [**–M** *suffix*] [**–p** *pathname*] – | *filename* . . .
xgettext **–h**

AVAILABILITY

SUNWloc

DESCRIPTION

xgettext is used to automate the creation of portable message files (**.po**). A **.po** file contains copies of "C" strings that are found in ANSI C source code in *filename* or the standard input if '–' is specified on the command line. The **.po** file can be used as input to the **msgfmt**(1) utility, which produces a binary form of the message file that can be used by application during run-time.

xgettext writes *msgid* strings from **gettext**(3I) calls in *filename* to the default output file **messages.po**. The default output file name can be changed by **–d** option. *msgid* strings in **dgettext( )** calls are written to the output file *domainname***.po** where *domainname* is the first parameter to the **dgettext( )** call.

By default, **xgettext** creates a **.po** file in the current working directory, and each entry is in the same order the strings are extracted from *filenames*. When the **–p** option is specified, the **.po** file is created in the *pathname* directory. An existing **.po** file is overwritten.

Duplicate *msgid*s are written to the **.po** file as comment lines. When the **–s** option is specified, the **.po** is sorted by the *msgid* string, and all duplicated *msgid*s are removed. All *msgstr* directives in the **.po** file are empty unless the **–m** option is used.

OPTIONS

–n	Add comment lines to the output file indicating file name and line number in the source file where each extracted string is encountered. These lines appear before each *msgid* in the following format: **#** **# File:** *filename***, line:** *line-number*
–s	Generate output sorted by *msgid*s with all duplicate *msgid*s removed.
–a	Extract all strings, not just those found in **gettext**(3I), and **dgettext ( )** calls. Only one **.po** file is created.
–c *comment-tag*	The comment block beginning with *comment-tag* as the first token of the comment block is added to the output **.po** file as # delimited comments. For multiple domains, **xgettext** directs comments and messages to the prevailing text domain.
–d *default-domain*	Rename default output file from **messages.po** to *default-domain* **.po**.
–j	Join messages with existing message files. If a **.po** file does not exist, it is created. If a **.po** file does exist, new messages are appended. Any duplicate **msgid**s are commented out in the resulting **.po** file. Domain directives in the existing **.po** file are ignored. Results not guaranteed if the existing message file has been edited.

–m *prefix*	Fill in the *msgstr* with *prefix*. This is useful for debugging purposes. To make *msgstr* identical to *msgid*, use an empty string ("") for *prefix*.
–M *suffix*	Fill in the *msgstr* with *suffix*. This is useful for debugging purposes.
–p *pathname*	Specify the directory where the output files will be placed. This option overrides the current working directory.
–x *exclude-file*	Specify a **.po** file that contains a list of *msgid*s that are not to be extracted from the input files. The format of *exclude-file* is identical to the **.po** file. However, only the *msgid* directive line in *exclude-file* is used. All other lines are simply ignored. The –**x** option can only be used with the –**a** option.
–h	Print a help message on the standard output.

SEE ALSO

msgfmt(1), **gettext**(3I)

NOTES

xgettext is not able to extract cast strings, for example ANSI C casts of literal strings to **(const char *)**. This is unnecessary anyway, since the prototypes in **<libintl.h>** already specify this type.

NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

xstr –c *filename* [**–v**] [**–l** *array*]
xstr [**–l** *array*]
xstr *filename* [**–v**] [**–l** *array*]

AVAILABILITY

SUNWcsu

DESCRIPTION

xstr maintains a file called **strings** into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, which are most useful if they are also read-only.

The command:

```
example% xstr –c filename
```

extracts the strings from the C source in name, replacing string references by expressions of the form **&xstr[***number***]** for some number. An appropriate declaration of **xstr** is prepended to the file. The resulting C text is placed in the file **x.c**, to then be compiled. The strings from this file are placed in the **strings** data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled, a file declaring the common **xstr** space called **xs.c** can be created by a command of the form:

```
example% xstr
```

This **xs.c** file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

xstr can also be used on a single file. A command:

```
example% xstr filename
```

creates files **x.c** and **xs.c** as before, without using or affecting any **strings** file in the same directory.

It may be useful to run **xstr** after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. **xstr** reads from the standard input when the argument '–' is given. An appropriate command sequence for running **xstr** after the C preprocessor is:

```
example% cc –E name.c | xstr –c –
example% cc –c x.c
example% mv x.o name.o
```

xstr does not touch the file **strings** unless new items are added; thus **make**(1S) can avoid remaking **xs.o** unless truly necessary.

OPTIONS

–c *filename*	Take C source text from *filename*.
–v	Verbose: display a progress report indicating where new or duplicate strings were found.
–l *array*	Specify the named *array* in program references to abstracted strings. The default array name is **xstr**.

FILES

strings	data base of strings
x.c	massaged C source
xs.c	C source for definition of array “xstr∗(rq
/tmp/xs∗	temp file when **xstr** *filename* doesn’t touch **strings**

SEE ALSO

make(1S)

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by **xstr** both strings will be placed in the data base, when just placing the longer one there would do.

NOTES

Be aware that **xstr** indiscriminately replaces all strings with expressions of the form **&xstr[***number***]** regardless of the way the original C code might have used the string. For example, you will encounter a problem with code that uses **sizeof( )** to determine the length of a literal string because **xstr** will replace the literal string with a pointer that most likely will have a different size than the string’s. To circumvent this problem:

- use **strlen( )** instead of **sizeof( )**; note that **sizeof( )** returns the size of the array (including the null byte at the end), whereas **strlen( )** doesn’t count the null byte. The equivalent of **sizeof("***xxx***")** really is **(strlen("***xxx***"))+1**.
- use **#define** for operands of **sizeof( )** and use the **define**’d version. **xstr** ignores **#define** statements. Make sure you run **xstr** on *filename* before you run it on the preprocessor.

You will also encounter a problem when declaring an initialized character array of the form

char x[] = "*xxx***";**

xstr will replace *xxx* with an expression of the form **&xstr[***number***]** which will not compile. To circumvent this problem, use **static char** ∗**x = "xxx"** instead of **static char x[] = "xxx"**.

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [**–vVdlt**] [**–Q** [**y** | **n**]] [**–p** *driver-file*] *filename*

DESCRIPTION

The **yacc** command converts a context-free grammar into a set of tables for a simple automaton that executes an LALR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a function **yyparse**( ). This program must be loaded with the lexical analyzer program, **yylex**( ), as well as **main**( ) and **yyerror**( ), an error handling routine. These routines must be supplied by the user; the **lex**(1) command is useful for creating lexical analyzers usable by **yacc**.

OPTIONS

–d	Generates the file **y.tab.h** with the **#define** statements that associate the **yacc** user-assigned ‘‘token codes’’ with the user-declared ‘‘token names.’’ This association allows source files other than **y.tab.c** to access the token codes.
–l	Specifies that the code produced in **y.tab.c** will not contain any **#line** constructs. This option should only be used after the grammar and the associated actions are fully debugged.
–t	Compiles runtime debugging code by default. Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. Whether or not the **–t** option is used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.
–v	Prepares the file **y.output**, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.
–V	Prints on the standard error output the version information for **yacc**.
-Q[**y** | **n**]	The **–Qy** option puts the version stamping information in **y.tab.c**. This allows you to know what version of **yacc** built the file. The **–Qn** option (the default) writes no version information.
–p *driver-file*	Allows you to specify the parser of your choice instead of **/usr/ccs/bin/yaccpar**. For example, you can specify:

```
yacc –p ~/myparser parser.y
```

ENVIRONMENT

The environment variables **LC_CTYPE** and **LANG** control the character classification throughout **yacc**. On entry to **yacc**, these environment variables are checked in the following order: **LC_CTYPE** and **LANG**. When a valid value is found, remaining environment variables for character classification are ignored. For example, a new setting for **LANG** does not override the current valid character classification rules of **LC_CTYPE**.

When none of the values are valid, the shell character classification defaults to the POSIX.1 C locale.

yacc can handle characters from EUC primary and supplementary codesets as one-token symbols. EUC codes may only be single character quoted terminal symbols. **yacc** expects **yylex**( ) to return a wide character (**wchar_t**) value for these one-token symbols.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.acts	temporary file
yacc.debug	temporary file
yacc.tmp	temporary file
yaccpar	parser prototype for C programs

SEE ALSO

lex(1)

The **yacc** chapter in the *Programming Utilities Guide* manual.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this instance is also reported.

NOTES

Because file names are fixed, at most one **yacc** process can be active in a given directory at a given time.

NAME

ypcat – print values in a NIS database

SYNOPSIS

ypcat [–**k**] [–**d** *ypdomain*] *mname*

AVAILABILITY

SUNWnisu

DESCRIPTION

The **ypcat** command prints out values in the NIS name service map specified by *mname*, which may be either a map name or a map nickname. Since **ypcat** uses the NIS network services, no NIS server is specified.

Refer to **ypfiles**(4) for an overview of the NIS name service.

OPTIONS

–**k**	Display the keys for those maps in which the values are null or the key is not part of the value. None of the maps derived from files that have an ASCII version in **/etc** fall into this class.
–**d** *ypdomain*	Specify a domain other than the default domain.

SEE ALSO

ypmatch(1), **ypfiles**(4)

NAME

ypmatch – print the value of one or more keys from a NIS map

SYNOPSIS

ypmatch [**–k**] [**–t**] [**–d** *domain*] *key* [*key*. . .] *mname*

ypmatch –x

AVAILABILITY

SUNWnisu

DESCRIPTION

ypmatch prints the values associated with one or more keys from the NIS's name services map specified by *mname*, which may be either a *mapname* or a map nickname (*mnames*).

Multiple keys can be specified; all keys will be searched for in the same map. The keys must be the same case and length. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

OPTIONS

–k	Before printing the value of a key, print the key itself, followed by a ':' (colon).
–t	This option inhibits map nickname translation.
–d *domain*	Specify a domain other than the default domain.
–x	Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypcat(1), **ypfiles**(4)

NAME

yppasswd – change your network password in the NIS database

SYNOPSIS

yppasswd [*username*]

AVAILABILITY

SUNWcsu

DESCRIPTION

yppasswd changes the network password associated with the user *username* in the Network Information Service (NIS) database. If the user has done a **keylogin**(1), and a publickey / secretkey pair exists for the user in the NIS **publickey.byname** map, **yppasswd** also re-encrypts the secretkey with the new password. The NIS password may be different from the local one on your own machine. Use **passwd**(1) to change the password information on the local machine, and **nispasswd**(1) to change the password information stored in Network Information Service, Version 3 database (NIS+).

yppasswd prompts for the old NIS password, and then for the new one. You must type in the old password correctly for the change to take effect. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long, if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough. Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

The NIS password daemon, **rpc.yppasswdd** must be running on your NIS server in order for the new password to take effect.

WARNINGS

Even after the user has successfully changed his or her password using this command, the subsequent **login**(1) using the new password will be successful only if the user's password and shadow information is obtained from NIS, (see **getpwnam**(3C), **getspnam**(3C), and **nsswitch.conf**(4)).

SEE ALSO

keylogin(1), **login**(1), **passwd**(1), **nispasswd**(1), **getpwnam**(3C), **getspnam**(3C), **secure_rpc**(3N), **nsswitch.conf**(4)

BUGS

The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

NAME

ypwhich – return name of NIS server or map master

SYNOPSIS

ypwhich [**–d** *domain*] [[**–t**] **–m** [*mname*] *hostname*]
ypwhich –x

AVAILABILITY

SUNWnisu

DESCRIPTION

ypwhich returns the name of the NIS server that supplies the NIS name services to a NIS client, or which is the master for a map. If invoked without arguments, it gives the NIS server for the local machine. If *hostname* is specified, that machine is queried to find out which NIS master it is using.

Refer to **ypfiles**(4) for an overview of the NIS name services.

OPTIONS

–d *domain*	Use *domain* instead of the default domain.
–t	This option inhibits map nickname translation.
–m *mname*	Find the master NIS server for a map. No *hostname* can be specified with **–m**. *mname* can be a mapname, or a nickname for a map. When *mname* is omitted, produce a list of available maps.
–x	Display the map nickname translation table.

SEE ALSO

ypfiles(4)

Index