Guía del desarrollador para la creación de paquetes de aplicaciones

Referencia: E38875–01
Enero de 2013

ORACLE

130224@25097

Contenido

Prefacio

La *Guía del desarrollador para la creación de paquetes de aplicaciones* contiene instrucciones paso a paso e información básica importante para el diseño, construcción y verificación de paquetes. Esta guía también incluye técnicas avanzadas que pueden ser útiles durante el proceso de creación de paquetes.

Nota – Esta versión de Oracle Solaris es compatible con sistemas que usen arquitecturas de las familias de procesadores SPARC y x86. Los sistemas compatibles aparecen en *Listas de compatibilidad del sistema operativo Oracle Solaris*. Este documento indica las diferencias de implementación entre los tipos de plataforma.

En este documento, estos términos relacionados con x86 significan lo siguiente:

- x86 hace referencia a la familia más grande de productos compatibles con x86 de 32 y 64 bits.
- x64 hace referencia específicamente a CPU compatibles con x86 de 64 bits.
- "x86 de 32 bits" destaca información específica de 32 bits acerca de sistemas basados en x86.

Para conocer cuáles son los sistemas admitidos, consulte *Listas de compatibilidad del sistema operativo Oracle Solaris*.

Quién debe utilizar este manual

Esta guía va dirigida a desarrolladores de aplicaciones cuyas responsabilidades incluyan el diseño y la construcción de paquetes.

Aunque una gran parte de esta guía está dirigida a desarrolladores de paquetes sin experiencia, también contiene información útil para desarrolladores de paquetes más experimentados.

Organización de esta guía

En la tabla siguiente se describen los capítulos de este manual.

Nombre del capítulo	Descripción del capítulo
Capítulo 1, “Diseño de un paquete”	Describe los componentes de los paquetes y los criterios de diseño de éstos. También describe secuencias de comandos, archivos y comandos relacionados.
Capítulo 2, “Creación de un paquete”	Describe las tareas y el proceso necesarios para construir un paquete. También proporciona instrucciones paso a paso para cada tarea.
Capítulo 3, “Mejora de las funciones de un paquete (tareas)”	Proporciona instrucciones paso a paso para agregar funciones optativas a un paquete.
Capítulo 4, “Verificación y transferencia de un paquete”	Describe cómo verificar la integridad de un paquete y cómo transferir un paquete a un medio de distribución.
Capítulo 5, “Casos prácticos de creación de paquetes”	Proporciona casos prácticos para crear paquetes.
Capítulo 6, “Técnicas avanzadas para la creación de paquetes”	Describe técnicas avanzadas para crear paquetes.
Glosario	Define los términos utilizados en esta guía.

Manuales relacionados

La documentación siguiente, disponible en distribuidores minoristas, puede proporcionar información de fondo adicional sobre la construcción de paquetes de System V.

- *Interfaz binaria de aplicaciones de System V*
- *Interfaz binaria de aplicaciones de System V: suplemento para procesadores SPARC*
- *Interfaz binaria de aplicaciones de System V: suplemento para procesadores Intel386*

Acceso a Oracle Support

Los clientes de Oracle tienen acceso a soporte electrónico por medio de My Oracle Support. Para obtener más información, visite `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` o, si tiene alguna discapacidad auditiva, visite `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs`.

Convenciones tipográficas

La siguiente tabla describe las convenciones tipográficas utilizadas en este manual.

TABLA P–1 Convenciones tipográficas

Tipos de letra	Descripción	Ejemplo
`AaBbCc123`	Los nombres de los comandos, los archivos, los directorios y los resultados que el equipo muestra en pantalla	Edite el archivo `.login`. Utilice el comando `ls -a` para mostrar todos los archivos. `nombre_sistema% tiene correo.`
`AaBbCc123`	Lo que se escribe, en contraposición con la salida del equipo en pantalla	`machine_name%` **`su`** `Contraseña:`
aabbcc123	Marcador de posición: sustituir por un valor o nombre real	El comando necesario para eliminar un archivo es `rm` *filename*.
AaBbCc123	Títulos de los manuales, términos nuevos y palabras destacables	Consulte el capítulo 6 de la *Guía del usuario*. Una *copia en caché* es aquella que se almacena localmente. *No* guarde el archivo. **Nota:** Algunos elementos destacados aparecen en negrita en línea.

Indicadores de los shells en los ejemplos de comandos

La tabla siguiente muestra los indicadores de sistema UNIX predeterminados y el indicador de superusuario de shells que se incluyen en los sistemas operativos Oracle Solaris. Tenga en cuenta que el indicador predeterminado del sistema que se muestra en los ejemplos de comandos varía según la versión de Oracle Solaris.

TABLA P–2 Indicadores de shell

Shell	Indicador
Shell Bash, shell Korn y shell Bourne	`$`
Shell Bash, shell Korn y shell Bourne para superusuario	`#`
Shell C	`machine_name%`

TABLA P–2 Indicadores de shell *(Continuación)*

Shell	Indicador
Shell C para superusuario	`machine_name#`

◆ ◆ ◆ CAPÍTULO 1

Diseño de un paquete

Antes de construir un paquete, debe saber qué archivos necesita crear y los comandos que debe ejecutar. También debe considerar los requisitos del software de la aplicación y las necesidades de los clientes. Los clientes son los administradores que instalarán el paquete. Este capítulo trata sobre los archivos, comandos y criterios que debe conocer y tener en cuenta antes de construir un paquete.

A continuación se indica la información contenida en este capítulo:

Dónde buscar las tareas de creación de paquetes

Use estos mapas de tareas para buscar las instrucciones detalladas de construcción y verificación de los paquetes.

¿Qué es un paquete?

El software de la aplicación se entrega en unidades llamadas *paquetes*. Un paquete es una colección de archivos y directorios necesarios para un producto de software. El desarrollador de la aplicación es quien diseña y construye normalmente el paquete, después de completar el desarrollo del código de la aplicación. Un producto de software se debe construir en uno o más paquetes para que se pueda transferir fácilmente a un medio de distribución. Posteriormente, el producto de software se puede producir de forma masiva y lo pueden instalar los administradores.

Un paquete es una colección de archivos y directorios con un formato definido. Este formato se ajusta a la interfaz binaria de aplicaciones (ABI), un suplemento a la Definición de interfaz de sistema V.

Componentes del paquete

Los componentes de un paquete se dividen en dos categorías.

- Los *objetos de paquetes* son los archivos de la aplicación que se deben instalar.
- Los *archivos de control* controlan cómo, dónde y si el paquete está instalado.

Los archivos de control también se dividen en dos categorías: *archivos de información* y *secuencias de comandos de instalación*. Se precisan algunos archivos de control. Algunos archivos de control son optativos.

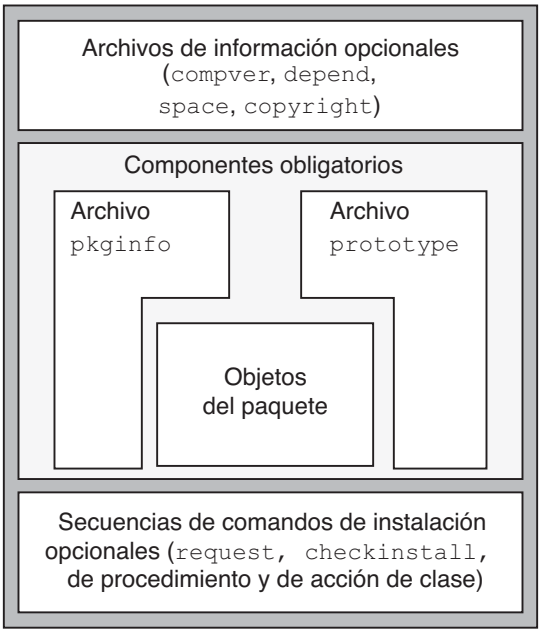
Para empaquetar las aplicaciones, en primer lugar debe crear los componentes necesarios, así como los componentes optativos que compongan el paquete. Posteriormente puede construir el paquete mediante el comando `pkgmk`.

Para construir un paquete, debe proporcionar lo siguiente:

- Objetos de paquetes (directorios y archivos de software de la aplicación)
- Dos archivos de información necesarios (los archivos `pkginfo` y `prototype`)
- Archivos de información opcionales
- Secuencias de comandos de instalación opcionales

En la figura siguiente se describe el contenido de un paquete.

FIGURA 1–1 El contenido de un paquete



Componentes del paquete necesarios

Debe crear los componentes siguientes antes de construir el paquete:

- Objetos del paquete

  Estos componentes componen la aplicación. Pueden estar formados por los elementos siguientes:

  - Archivos (archivos ejecutables o archivos de datos)
  - Directorios
  - Conducciones con nombre
  - Enlaces
  - Dispositivos

- El archivo `pkginfo`

  El archivo `pkginfo` es un archivo de información de paquetes necesario que define los valores de los parámetros. Los valores de los parámetros incluyen la abreviatura de los paquetes, el nombre completo del paquete y su arquitectura. Para obtener más información, consulte "Creación de un archivo `pkginfo`" en la página 26 y la página de comando man `pkginfo(4)`.

Nota – Hay dos páginas de comando man `pkginfo(1)` La primera página de comando man describe un comando de una sección que muestra información sobre paquetes instalados. La segunda página de comando man describe un archivo de 4 secciones que describe las características de un paquete. Al acceder a las páginas de comando man, asegúrese de especificar la sección de la pagina de comando man aplicable. Por ejemplo: `man -s 4 pkginfo`.

- El archivo `prototype`

  El archivo `prototype` es un archivo de información de paquetes necesario que enumera los componentes del paquete. Hay una entrada para cada objeto de paquete, archivo de información y secuencia de comandos de instalación. Una entrada consiste en varios campos de información que describen cada componente, incluida su ubicación, atributos y tipo de archivo. Para obtener más información, consulte "Creación de un archivo `prototype`" en la página 31 y la página de comando man `prototype(4)`.

Componentes del paquete opcionales

Archivos de información del paquete

Puede incluir cuatro archivos de información de paquetes optativos en el paquete:

- El archivo `compver`

  Define las versiones anteriores del paquete que son compatibles con esta versión del paquete.

- El archivo `depend`

  Indica otros paquetes que tienen una relación especial con el paquete.

- El archivo `space`

  Define requisitos de espacio en el disco para el entorno de destino, más allá de lo que necesitan los objetos definidos en el archivo `prototype`. Por ejemplo, puede que se necesite espacio adicional para archivos que se crean dinámicamente en el tiempo de la instalación.

- El archivo `copyright`

  Define el texto de un mensaje de copyright que aparece en el tiempo de la instalación del paquete.

Cada archivo de información del paquete debe tener una entrada en el archivo `prototype`. Consulte "Creación de archivos de información" en la página 52 para obtener más información sobre la creación de estos archivos.

Secuencias de comandos de instalación del paquete

Las secuencias de comandos de instalación no son necesarias. Sin embargo, puede proporcionar secuencias de comandos que llevan a cabo acciones personalizadas durante la instalación del paquete. Una secuencia de comandos de instalación tiene las características siguientes:

- La secuencia de comandos se compone de comandos shell Bourne.
- Los permisos de archivos de las secuencias de comandos deben establecerse en 0644.
- La secuencia de comandos no debe contener el identificador de shell (`#! /bin/sh`).

Los cuatro tipos de secuencias de comandos son los siguientes:

- La secuencia de comandos `request`

  La secuencia de comandos `request` solicita entrada de información del administrador que instala el paquete.
- La secuencia de comandos `checkinstall`

  La secuencia de comandos `checkinstall` lleva a cabo una verificación especial del sistema de archivos.

  Nota – La secuencia de comandos `checkinstall` sólo está disponible con Solaris 2.5 u otras versiones compatibles.

- Secuencias de comandos de procedimientos

  Las *secuencias de comandos de procedimientos* definen las acciones que tienen lugar en momentos concretos durante la instalación y eliminación de paquetes. Puede crear cuatro secuencias de comandos de procedimientos con estos nombres predefinidos: `preinstall`, `postinstall`, `preremove` y `postremove`.
- Secuencias de comandos de acción de clase

  Las *secuencias de comandos de acción de clase* definen un conjunto de acciones que se deben llevar a cabo en un grupo de objetos.

Consulte "Creación de secuencias de comandos de instalación" en la página 58 para obtener más información sobre las secuencias de comandos de instalación.

Consideraciones antes de construir un paquete

Antes de construir un paquete, debe decidir si el producto consistirá en uno o más paquetes. Tenga en cuenta que muchos paquetes pequeños necesitan más tiempo para instalarse que un paquete grande. Aunque la creación de un único paquete es una buena idea, no siempre es posible hacerlo. Si decide construir más de un paquete, debe determinar cómo segmentar el código de la aplicación. Esta sección proporciona una lista de criterios que usar al planificar la construcción de un paquete.

Muchos de los criterios de creación de paquetes se presentan como alternativas entre ellos. La satisfacción de todos los requisitos por igual es a menudo difícil. Estos criterios se presentan en orden de importancia. Sin embargo, esta secuencia está pensada para que sirva como guía flexible, según las circunstancias. Aunque cada criterio es importante, depende de usted optimizar estos requisitos para producir un buen conjunto de paquetes.

Si desea conocer más ideas de diseño, consulte el Capítulo 6, "Técnicas avanzadas para la creación de paquetes".

Creación de paquetes de instalación remota

Todos los paquetes deben ser de *instalación remota*. Por instalación remota se entiende que el administrador puede intentar la instalación del paquete en un sistema cliente, no necesariamente en el sistema de archivos raíz (`/`) donde se ejecuta el comando `pkgadd`.

Optimización de configuraciones de cliente-servidor

Considere los diversos tipos de configuraciones del software del sistema (por ejemplo, servidor y sistema autónomo) al distribuir los paquetes. Un buen diseño de creación de paquetes divide los archivos afectados para optimizar la instalación de cada tipo de configuración. Por ejemplo el contenido de los sistemas de archivos raíz (`/`) y `/usr` se debe segmentar para que las configuraciones de servidor se puedan admitir fácilmente.

Paquetes por restricciones funcionales

Los paquetes deben ser autónomos e identificarse de forma distintiva con un conjunto de funciones. Por ejemplo, un paquete que contiene UFS debe contener todas las utilidades de UFS y limitarse solamente a los binarios UFS.

Los paquetes se deben organizar desde el punto de vista del cliente en unidades funcionales.

Paquete con restricciones por derechos de autor

Coloque el código que precise el pago de derechos de autor debido a acuerdos contractuales en un paquete o grupo de paquetes específico. No disperse el código en más paquetes de los necesarios.

Paquete por dependencias del sistema

Conserve los binarios dependientes del sistema en paquetes dedicados. Por ejemplo, el código del núcleo debe estar en un paquete específico, con cada arquitectura de implementación constituida por una instancia de paquete distinta. Esta regla también se aplica a binarios para arquitecturas diferentes. Por ejemplo, los binarios para un sistema SPARC estarían en un paquete y los binarios para un sistema x86 estarían en otro.

Supresión de la superposición en los paquetes

Al construir los paquetes, suprima los archivos duplicados siempre que sea posible. La duplicación innecesaria de archivos provoca dificultades con las versiones y la asistencia técnica. Si el producto tiene varios paquetes, compare varias veces su contenido para buscar los archivos duplicados.

Paquete con restricciones en la ubicación

Los elementos específicos de la localización deben estar en su propio paquete. Un modelo de creación de paquetes ideal tendría las localizaciones de un producto distribuidas como un paquete por cada configuración regional. Desafortunadamente, en algunos casos, las restricciones organizativas entran en conflicto con los criterios de restricciones de productos y funcionales.

Los valores predeterminados internacionales también se pueden entregar en un paquete. Este diseño aísla los archivos que son necesarios para los cambios en la localización y estandariza el formato de distribución de los paquetes de localización.

Secuencias de comandos, archivos y comandos en la creación de paquetes

Esta sección describe los comandos, archivos y secuencias de comandos que puede usar cuando manipule los paquetes. Se describen en las páginas de comando man y de forma detallada en este manual, en relación con la tarea específica que desempeñan.

La tabla siguiente muestra los comandos que le ayudarán a construir, verificar, instalar y obtener información sobre un paquete.

TABLA 1–1 Comandos de empaquetado

Tarea	Comando/página del comando man	Descripción	Para obtener más información
Crear paquetes	`pkgproto`(1)	Genera un archivo `prototype` para la entrada de información al comando `pkgmk`	"Ejemplo: creación de un archivo `prototype` con el comando `pkgproto`" en la página 38
`pkgmk`(1)	Crea un paquete instalable	"Creación de un paquete" en la página 45	
Instalar, suprimir y transferir paquetes	`pkgadd`(1M)	Instala un paquete de software en un sistema	"Instalación de paquetes de software" en la página 90
`pkgask`(1M)	Guarda respuestas a una secuencia de comandos `request`	"Normativa de diseño para secuencias de comandos `request`" en la página 64	
`pkgtrans`(1)	Copia paquetes en un medio de distribución	"Transferencia de un paquete a un medio de distribución" en la página 99	
`pkgrm`(1M)	Suprime un paquete de un sistema	"Eliminación de un paquete" en la página 99	
Obtener información sobre paquetes	`pkgchk`(1M)	Verifica la integridad de un paquete de software	"Verificación de la integridad de un paquete" en la página 92
`pkginfo`(1)	Muestra información de un paquete de software	"El comando `pkginfo`" en la página 95	
`pkgparam`(1)	Muestra los valores de los parámetros del paquete	"El comando `pkgparam`" en la página 94	
Modificar paquetes instalados	`installf`(1M)	Incorpora un nuevo objeto de paquete en un paquete ya instalado	"Normativa de diseño para secuencias de comandos de procedimientos" en la página 70 y Capítulo 5, "Casos prácticos de creación de paquetes"
`removef`(1M)	Suprime un objeto de paquete de un paquete ya instalado	"Normativa de diseño para secuencias de comandos de procedimientos" en la página 70	

La tabla siguiente muestra los archivos de información que le ayudarán a construir un paquete.

TABLA 1–2 Archivos de información del paquete

Archivo	Descripción	Para obtener más información
admin(4)	Archivo predeterminado de instalación de paquetes	"El archivo administrativo predeterminado" en la página 132
compver(4)	Archivo de compatibilidad del paquete	"Definición de dependencias de paquetes" en la página 53
copyright(4)	Archivo de información de copyright del paquete	"Escritura de un mensaje de copyright" en la página 55
depend(4)	Archivo de dependencias del paquete	"Definición de dependencias de paquetes" en la página 53
pkginfo(4)	Archivo de características del paquete	"Creación de un archivo pkginfo" en la página 26
pkgmap(4)	Archivo de descripción de contenido del paquete	"El archivo pkgmap" en la página 45
prototype(4)	Archivo de información del paquete	"Creación de un archivo prototype" en la página 31
space(4)	Archivo de requisitos de espacio en el disco del paquete	"Reserva del espacio adicional en un sistema de destino" en la página 57

La tabla siguiente describe secuencias de comandos de instalación optativas que puede escribir que pueden afectan según cómo esté instalado un paquete.

TABLA 1–3 Secuencias de comandos de instalación del paquete

Secuencia de comandos	Descripción	Para obtener más información
request	Solicita información del instalador	"Escritura de una secuencia de comandos request" en la página 64
checkinstall	Reúne datos del sistema de archivos	"Reunión de datos del sistema de archivos con la secuencia de comandos checkinstall" en la página 66
preinstall	Lleva a cabo los requisitos de instalación personalizados antes instalar la clase	"Escritura de secuencias de comandos de procedimientos" en la página 69
postinstall	Lleva a cabo los requisitos de instalación personalizados después de instalar todos los volúmenes	"Escritura de secuencias de comandos de procedimientos" en la página 69

TABLA 1–3 Secuencias de comandos de instalación del paquete *(Continuación)*

Secuencia de comandos	Descripción	Para obtener más información
`preremove`	Lleva a cabo los requisitos de eliminación personalizados antes de suprimir la clase	"Escritura de secuencias de comandos de procedimientos" en la página 69
`postremove`	Lleva a cabo los requisitos de eliminación personalizados después de que todas las clases se hayan suprimido	"Escritura de secuencias de comandos de procedimientos" en la página 69
Acción de clase	Lleva a cabo una serie de acciones en un grupo de objetos específico	"Escritura de secuencias de comandos de acción de clase" en la página 71

◆ ◆ ◆ CAPÍTULO 2

Creación de un paquete

En este capítulo se describen el proceso y las tareas implicadas en la construcción de un paquete. Algunas de estas tareas son necesarias. Algunas de estas tareas son optativas. Las tareas necesarias se tratan de forma detallada en este capítulo. Para obtener más información sobre las tareas optativas que permiten agregar funciones al paquete, consulte el Capítulo 3, "Mejora de las funciones de un paquete (tareas)", y el Capítulo 6, "Técnicas avanzadas para la creación de paquetes".

A continuación se indica la información contenida en este capítulo:

El proceso de construcción de un paquete (mapa de tareas)

Tabla 2–1 describe un proceso que puede seguir a la hora de construir paquetes, especialmente si no tiene experiencia en construirlos. Aunque no es obligatorio completar las cuatro primeras tareas en el orden exacto en que aparecen, la experiencia de construir paquetes será más sencilla si lo hace. Cuando sea un diseñador de paquetes experimentado, puede adaptar la secuencia de estas tareas a sus preferencias.

Como diseñador de paquetes experimentado, puede automatizar el proceso de construcción de paquetes mediante el uso del comando `make` y los makefiles. Para obtener más información, consulte la página de comando man `make(1S)`.

TABLA 2–1 El proceso de construcción de un paquete (mapa de tareas)

Tarea	Descripción	Para obtener instrucciones
1. Crear un archivo `pkginfo`	Cree el archivo `pkginfo` para describir las características del paquete.	"Cómo crear un archivo `pkginfo`" en la página 29
2. Organizar el contenido del paquete	Ordene los componentes del paquete en una estructura de directorios jerárquica.	"Organización del contenido de un paquete" en la página 30
3. (optativo) Crear archivos de información	Defina dependencias de paquetes, incluido un mensaje de copyright y reserve espacio adicional en el sistema de destino.	Capítulo 3, "Mejora de las funciones de un paquete (tareas)"
4. (optativo) Crear secuencias de comandos de instalación	Personalice los procesos de instalación y eliminación del paquete.	Capítulo 3, "Mejora de las funciones de un paquete (tareas)"
5. Crear un archivo `prototype`	Describa el objeto del paquete en un archivo `prototype`.	"Creación de un archivo `prototype`" en la página 31
6. Construir el paquete	Construya el paquete mediante el comando `pkgmk`.	"Creación de un paquete" en la página 45
7. Verificar y transferir el paquete	Verifique la integridad del paquete antes de copiarlo en un medio de distribución.	Capítulo 4, "Verificación y transferencia de un paquete"

Variables de entorno de paquetes

Puede usar variables en los archivos de información necesarios, `pkginfo` y `prototype`. También puede usar una opción con el comando `pkgmk`, que se utiliza para construir un paquete. Todos estos archivos y comandos se tratan en este capítulo; se ofrece asimismo más información contextual sobre variables. Sin embargo, cuando comience a construir el paquete, debe comprender los diferentes tipos de variables y cómo pueden afectar a la creación satisfactoria de un paquete.

Hay dos tipos de variables:

- Construir variables

  Las variables de construcción comienzan por minúscula y se evalúan en el *tiempo de construcción*, ya que el paquete se construye con el comando `pkgmk`.

- Variables de instalación

  Las variables de instalación comienzan por mayúscula y se evalúan en el *tiempo de instalación*, ya que el paquete se instala con el comando `pkgadd`.

Normas generales sobre el uso de variables de entorno

En el archivo `pkginfo`, una definición de variable toma el formato de *PARAM=value*, donde la primera letra de *PARAM* está en mayúsculas. Estas variables se evalúan sólo en el tiempo de la instalación. Si alguna de estas variables no se puede evaluar, el comando `pkgadd` se cancela con un error.

En el archivo `prototype`, una definición de variable puede tomar el formato `!`*PARAM=value* o `$`*variable*. Tanto *PARAM* como *variable* puede comenzar por mayúsculas o minúsculas. Sólo se evalúan las variables cuyos valores sean conocidos en el tiempo de la construcción. Si *PARAM* o *variable* es una variable de construcción o de instalación cuyo valor no sea conocido en el tiempo de la construcción, el comando `pkgmk` se cancela con un error.

También puede incluir *PARAM=value* como opción para el comando `pkgmk`. Esta opción trabaja del mismo modo que en el archivo `prototype`, excepto que su ámbito es global en todo el paquete. La definición `!`*PARAM=value* en un archivo `prototype` es local para ese archivo y la parte del paquete que define.

Si *PARAM* es una variable de instalación y *variable* es una variable de instalación o de construcción con un valor conocido, el comando `pkgmk` inserta la definición en el archivo `pkginfo` de forma que la definición estará disponible en el tiempo de la instalación. Sin embargo, el comando `pkgmk` no evalúa las *PARAM* que se encuentren en nombres de ruta especificados en el archivo `prototype`.

Resumen de variables de entorno de paquetes

La tabla siguiente resume la ubicación, el ámbito y los formatos de especificación de variables.

TABLA 2–2 Resumen de variables de entorno de paquetes

Donde se define la variable	Formato de definición de variable	Tipo de variable que se define	Cuando la variable se evalúa	Donde se evalúa la variable	Elementos por los que se puede sustituir la variable
Archivo `pkginfo`	*PARAM=value*	Construcción	Ignorado en el tiempo de la construcción	N/A	Ninguno
Instalación	Tiempo de la instalación	En el archivo `pkgmap`	*owner*, *group*, *path* o destino de vínculo		
Archivo `prototype`	`!`*PARAM=value*	Construcción	Tiempo de construcción	En el archivo `prototype` y cualquier archivo que se incluya	*mode*, *owner*, *group* o *path*

TABLA 2–2 Resumen de variables de entorno de paquetes *(Continuación)*

Donde se define la variable	Formato de definición de variable	Tipo de variable que se define	Cuando la variable se evalúa	Donde se evalúa la variable	Elementos por los que se puede sustituir la variable
Instalación	Tiempo de construcción	En el archivo `prototype` y cualquier archivo que se incluya	Comandos `!search` y `!command` solamente		
Línea de comandos de `pkgmk`	*PARAM=value*	Construcción	Tiempo de construcción	En el archivo `prototype`	*mode*, *owner*, *group* o *path*
Instalación	Tiempo de construcción	En el archivo `prototype`	Comando `!search` solamente		
Tiempo de la instalación	En el archivo `pkgmap`	*owner*, *group*, *path* o destino de vínculo			

Creación de un archivo `pkginfo`

`pkginfo` es un archivo ASCII que describe las características de un paquete, así como la información que ayuda a controlar el flujo de la instalación.

Cada entrada del archivo `pkginfo` es una línea que establece el valor de un parámetro mediante el formato *PARAM=value*. *PARAM* puede ser cualquiera de los parámetros estándar que se describen en la página de comando man `pkginfo(4)` No hay un orden preciso en el que los parámetros se deban especificar.

Nota – Cada *valor* se puede incluir entre comillas simples o dobles (por ejemplo, '*valor*' o "*valor*"). Si *valor* contiene caracteres que se consideren especiales para un entorno de shell, debe usar comillas. Los ejemplos y los casos prácticos de este manual no utilizan comillas. Consulte la página de comando man `pkginfo(4)` para ver algún ejemplo que utilice comillas dobles.

También puede asignar un valor a sus propios parámetros del paquete para crearlos en el archivo `pkginfo`. Los parámetros deben comenzar por una letra en mayúsculas seguida de letras en mayúsculas o minúsculas. Una letra en mayúsculas indica que el parámetro (variable) se evaluará en el tiempo de la instalación (en contraposición al tiempo de la construcción). Para obtener más información sobre la diferencia entre las variables de instalación y de construcción, consulte "Variables de entorno de paquetes" en la página 24.

Nota – Los espacios en blanco que haya tras el valor de un parámetro se ignoran.

Debe definir estos cinco parámetros en un archivo pkginfo: PKG, NAME, ARCH, VERSION y CATEGORY. El software inserta automáticamente los parámetros PATH, PKGINST y INSTDATE cuando se construye el paquete. No modifique estos ocho parámetros. Para obtener información sobre los parámetros restantes, consulte la página de comando man pkginfo(4).

Definición de la instancia de un paquete

El mismo paquete puede tener versiones diferentes, ser compatible con arquitecturas diferentes o ambas. Cada variación de un paquete se conoce como *instancia de paquete*. Una instancia de paquete está determinada por la combinación de las definiciones de los parámetros PKG, ARCH y VERSION en el archivo pkginfo.

El comando pkgadd asigna un *identificador de paquete* a cada instancia de paquete en el tiempo de la instalación. El identificador es la abreviatura del paquete con un sufijo numérico, por ejemplo SUNWadm.2. Este identificador distingue una instancia de cualquier otro paquete, incluidas las instancias del mismo paquete.

Definición de la abreviatura de un paquete (PKG)

Una *abreviatura de paquete* es un nombre más corto de un paquete definido por el parámetro PKG en el archivo pkginfo. La abreviatura de un paquete debe tener estas características:

- La abreviatura debe componerse de caracteres alfanuméricos. El primer carácter no puede ser un número.
- La abreviatura no puede superar los 32 caracteres de longitud.
- La abreviatura no puede ser una de las reservadas: install, new o all.

Nota – Los cuatro primeros caracteres deben ser exclusivos de su empresa. Por ejemplo, los paquetes construidos por Sun Microsystems tendrán todos "SUNW" como los cuatro primeros caracteres de la abreviatura de sus paquetes.

Un ejemplo de entrada de abreviatura de paquete en un pkginfo es PKG=SUNWcadap.

Especificación de una arquitectura de paquetes (ARCH)

El parámetro ARCH en el archivo pkginfo identifica qué arquitecturas se asocian al paquete. El nombre de la arquitectura tiene un máximo de 16 caracteres alfanuméricos. Si un paquete se ha asociado a más de una arquitectura, especifíquelas en una lista separadas por comas.

Éste es un ejemplo de una especificación de arquitectura de paquetes en un archivo pkginfo:

```
ARCH=sparc
```

Especificación de una arquitectura de conjunto de instrucciones de paquetes (SUNW_ISA)

El parámetro SUNW_ISA del archivo pkginfo identifica qué arquitectura de conjunto de instrucciones está asociada a un paquete de Sun Microsystems. Los valores son los siguientes:

- sparcv9, para un paquete que contiene objetos de 64 bits
- sparc para un paquete que contiene objetos de 32 bits

Por ejemplo, el valor SUNW_ISA de un archivo pkginfo para un paquete que contiene objetos de 64 bits tendría este aspecto:

```
SUNW_ISA=sparcv9
```

Si no se ha configurado SUNW_ISA, la arquitectura predeterminada del conjunto de instrucciones se configura con el valor del parámetro ARCH.

Especificación de la versión de un paquete (VERSION)

El parámetro VERSION del archivo pkginfo identifica la versión del paquete. La versión tiene un máximo de 256 caracteres ASCII y no puede comenzar con un paréntesis izquierdo.

Ésta es una versión de ejemplo de una especificación en un archivo pkginfo:

```
VERSION=release 1.0
```

Definición del nombre de un paquete (NAME)

Un *nombre de paquete* es el nombre completo del paquete, definido por el parámetro NAME en el archivo pkginfo.

Debido a que los administradores del sistema usan con frecuencia nombres de paquetes para determinar si un paquete debe instalarse, es importante escribir nombres de paquetes claros, concisos y completos. Los nombres de paquetes deben satisfacer estos criterios:

- Indique cuándo se necesita un paquete (por ejemplo, para proporcionar determinados comandos o funciones, o bien indicar si el paquete es necesario para un hardware específico).
- Indique para qué se usa el paquete (por ejemplo, el desarrollo de controladores de dispositivos).
- Incluya una descripción nemotécnica de la abreviatura del paquete, mediante palabras clave que indiquen que la abreviatura es una forma corta de la descripción. Por ejemplo, el nombre de la abreviatura del paquete SUNWbnuu es “Basic Networking UUCP Utilities, (Usr)” .
- Nombre la partición en la que el paquete está instalado.

- Use los términos de forma coherente con el significado en el sector.
- Benefíciese del límite de 256 caracteres.

A continuación puede ver un nombre de paquete de ejemplo definido en un archivo pkginfo:

```
NAME=Chip designers need CAD application software to design
abc chips.  Runs only on xyz hardware and is installed in the
usr partition.
```

Definición de una categoría de paquetes (CATEGORY)

El parámetro CATEGORY del archivo pkginfo especifica a qué categorías pertenece un catálogo. Como mínimo, un paquete debe pertenecer a la categoría system o application. Los nombres de las categorías se componen de caracteres alfanuméricos. Los nombres de categorías tienen una longitud máxima de 16 caracteres y distinguen entre mayúsculas y minúsculas.

Si un paquete pertenece a más de una categoría, especifíquelas en una lista, separadas por comas.

A continuación puede ver una especificación CATEGORY de ejemplo en un archivo pkginfo:

```
CATEGORY=system
```

▼ Cómo crear un archivo pkginfo

1. **Cree un archivo llamado pkginfo con su editor de texto preferido.**

   Puede crear este archivo en cualquier ubicación del sistema.

2. **Edite el archivo y defina los cinco parámetros necesarios.**

   Los cinco parámetros necesarios son: PKG, NAME, ARCH, VERSION y CATEGORY. Para obtener más información sobre estos parámetros, consulte "Creación de un archivo pkginfo" en la página 26.

3. **Agregue parámetros optativos al archivo.**

   Cree sus propios parámetros o consulte la página de comando man pkginfo(4) para obtener información sobre los parámetros estándar.

4. **Guarde los cambios y salga del editor.**

Ejemplo 2–1 Creación de un archivo pkginfo

Este ejemplo muestra el contenido de un archivo pkginfo válido, con los cinco parámetros necesarios definidos, así como el parámetro BASEDIR. El parámetro BASEDIR se trata más detalladamente en "El campo *path*" en la página 33.

```
PKG=SUNWcadap
NAME=Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.
ARCH=sparc
VERSION=release 1.0
CATEGORY=system
BASEDIR=/opt
```

Véase también Consulte "Cómo organizar el contenido de un paquete" en la página 30.

Organización del contenido de un paquete

Organice los objetos del paquete en una estructura jerárquica de directorios que refleje la estructura que los objetos del paquete tendrán en el sistema de destino después de la instalación. Si lleva a cabo este paso antes de crear un archivo `prototype`, puede ahorrar algo de tiempo y esfuerzo a la hora de crear ese archivo.

▼ Cómo organizar el contenido de un paquete

1 **Determine cuántos paquetes necesita crear y qué objetos se incluirán en cada paquete.**

Para obtener ayuda sobre la ejecución de este paso, consulte "Consideraciones antes de construir un paquete" en la página 17.

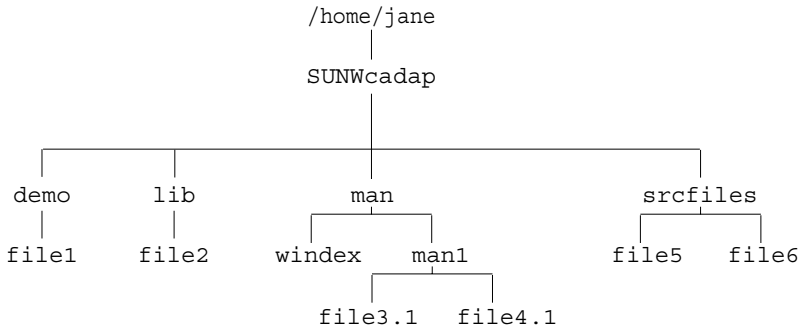
2 **Cree un directorio para cada paquete que necesite construir.**

Puede crear este directorio en cualquier lugar del sistema y nombrarlo como desee. Los ejemplos de este capítulo asumen que un directorio del paquete tiene el mismo nombre que la abreviatura del paquete.

```
$ cd /home/jane
$ mkdir SUNWcadap
```

3 **Organice los objetos de cada paquete en una estructura de directorios bajo su correspondiente directorio de paquetes. La estructura de directorios debe reflejar la estructura que los objetos del paquete tendrán en el sistema de destino.**

Por ejemplo, el paquete de la aplicación CAD, SUNWcadap, necesita la siguiente estructura de directorios.



4 **Decida dónde conservará los archivos de información. Si procede, cree un directorio para conservar los archivos en una ubicación.**

En este caso se asume que el archivo pkginfo de ejemplo de "Cómo crear un archivo pkginfo" en la página 29 se creó en el directorio de inicio de Jane.

```
$ cd /home/jane
$ mkdir InfoFiles
$ mv pkginfo InfoFiles
```

Véase también Consulte "Cómo crear un archivo prototype mediante el comando pkgproto" en la página 43.

Creación de un archivo prototype

prototype es un archivo ASCII utilizado para especificar información sobre los objetos de un paquete. Cada entrada del archivo prototype describe un único objeto, como un archivo de datos, un directorio, un archivo de origen o un objeto ejecutable. Las entradas de un archivo prototype constan de varios campos de información, separados por espacios en blanco. Observe que los campos *deben* aparecer en un orden concreto. Las líneas de comentarios comienzan por el signo de almohadilla (#) y se ignoran.

Puede crear un archivo prototype con un editor de texto o mediante el comando pkgproto. Cuando cree este archivo por primera vez, es probablemente más fácil hacerlo con el comando pkgproto, porque lo crea basándose en la jerarquía de directorios que haya creado anteriormente. Si no ha organizado los archivos tal como se describe en "Organización del contenido de un paquete" en la página 30, tiene la pesada tarea de crear el archivo prototype a partir de cero con el editor de textos que desee.

Formato del archivo prototype

A continuación puede ver el formato de cada línea en el archivo prototype:

partftypeclasspathmajorminormodeownergroup

part	Es un campo numérico optativo que le permite agrupar objetos de paquetes en partes. El valor predeterminado es parte 1.
ftype	Es un campo de un carácter que especifica el tipo de objeto. Consulte "El campo ftype" en la página 32.
class	Es la clase de instalación a la que pertenece el objeto. Consulte "El campo *class*" en la página 33.
path	Es un nombre de ruta absoluta o relativa que indica dónde se ubicará el objeto del paquete en el sistema de destino. Consulte "El campo *path*" en la página 33.
major	Es el número de dispositivo principal para dispositivos especiales de caracteres o bloques.
minor	Es el número de dispositivo secundario para dispositivos especiales de caracteres o bloques.
mode	Es el modo octal del objeto (por ejemplo, 0644). Consulte "El campo *mode*" en la página 36.
owner	Es el propietario del objeto (por ejemplo, bin o root). Consulte "El campo *owner*" en la página 36.
group	Es el grupo al que pertenecen los objetos (por ejemplo, bin o sys). Consulte "El campo *group*" en la página 37.

Por lo general, sólo se definen los campos *ftype*, *class*, *path*, *mode*, *owner* y *group*. Estos campos se describen en las secciones siguientes. Consulte la página de comando man prototype(4) para obtener información adicional sobre estos campos.

El campo ftype

ftype, o tipo de archivo, es un campo de un carácter que especifica el tipo de objeto de un paquete. Los tipos de archivos válidos se describen en la tabla siguiente

TABLA 2–3 Tipos de archivos válidos en el archivo prototype

Valor de campo de tipo de archivo	Descripción del tipo de archivo
f	Archivo ejecutable estándar o archivo de datos

TABLA 2–3 Tipos de archivos válidos en el archivo `prototype` *(Continuación)*

Valor de campo de tipo de archivo	Descripción del tipo de archivo
`e`	Archivo que se debe editar tras la instalación o la eliminación (lo pueden compartir varios paquetes)
`v`	Archivo volátil (cuyo contenido se espera que cambie, como un archivo de registro)
`d`	Directorio
`x`	Directorio exclusivo accesible sólo por este paquete (puede contener información de la base de datos o registros sin registrar)
`l`	Archivo vinculado
`p`	Conducción con nombre
`c`	Dispositivo especial de caracteres
`b`	Dispositivo especial de bloques
`i`	Archivo de información o secuencia de comandos de instalación
`s`	Enlace simbólico

El campo *class*

Los nombres del campo *clase* muestran la clase a la que pertenece un objeto. El uso de clases es una función optativa de diseño de paquetes. Esta función se trata detalladamente en "Escritura de secuencias de comandos de acción de clase" en la página 71.

Si no usa clases, un objeto pertenece a la clase `none`. Si ejecuta el comando `pkgmk` para construir el paquete, el comando inserta el parámetro `CLASSES=none` en el archivo `pkginfo`. Los archivos del tipo `i` deben tener un campo *class* en blanco.

El campo *path*

El campo *path* se utiliza para definir dónde residirá el objeto de paquete en el sistema de destino. Puede indicar la ubicación con un nombre de ruta absoluta (por ejemplo, `/usr/bin/mail`) o un nombre de ruta relativa (por ejemplo, `bin/mail`). El uso de un nombre de ruta absoluta significa que la ubicación del objeto en el sistema de destino la define el paquete y no se puede cambiar. Los objetos del paquete con nombres de ruta relativa indican que el objeto es *reubicable*.

Un *objeto reubicable* no necesita una ubicación de ruta absoluta en el sistema de destino. Por el contrario, la ubicación del objeto se determina durante el proceso de instalación.

Todos o algunos de los objetos de un paquete se pueden definir como reubicables. Antes de escribir secuencias de comandos de instalación o crear el archivo `prototype`, decida si los objetos del paquete tendrán una ubicación fija (como secuencias de comandos de inicio en `/etc`) o si serán reubicables.

Hay dos tipos de objetos reubicables: *reubicables colectivamente* y *reubicables individualmente*.

Objetos reubicables colectivamente

Los objetos reubicables colectivamente se sitúan en relación a una base de instalación común llamada *directorio base*. Un directorio base se define en el archivo `pkginfo`, mediante el parámetro `BASEDIR`. Por ejemplo, un objeto reubicable del archivo `prototype` con el nombre `tests/generic` necesita que el archivo `pkginfo` defina el parámetro `BASEDIR` predeterminado. Por ejemplo:

```
BASEDIR=/opt
```

Este ejemplo significa que cuando se instala el objeto, se ubicará en el directorio `/opt/tests/generic`.

Nota – El directorio `/opt` es el único directorio al que se puede entregar software que no forma parte del software de Oracle Solaris base.

Use objetos reubicables colectivamente siempre que sea posible. En general, la parte principal de un paquete se puede reubicar con unos cuantos archivos (como los archivos en `/etc` o `/var`) especificados como absolutos. Sin embargo, si un paquete contiene muchas reubicaciones diferentes, considere la opción de dividir el paquete en diversos paquetes con diferentes valores `BASEDIR` en sus archivos `pkginfo`.

Objetos reubicables individualmente

Los objetos reubicables individualmente no se restringen a la misma ubicación del directorio como objetos reubicables colectivamente. Para definir un objeto reubicable individualmente, debe especificar una variable de instalación en el campo *path* del archivo `prototype`. Después de especificar la variable de instalación, cree una secuencia de comandos `request` para solicitar al instalador el directorio base reubicable, o bien una secuencia de comandos `checkinstall` para determinar el nombre de ruta a partir de los datos del sistema de archivos. Para obtener más información sobre las secuencias de comandos `request`, consulte "Escritura de una secuencia de comandos `request`" en la página 64; para obtener más información sobre las secuencias de comandos `checkinstall`, consulte "Cómo reunir datos del sistema de archivos" en la página 68.

Precaución – Los objetos reubicables individualmente son difíciles de administrar. El uso de objetos reubicables individualmente puede provocar que haya componentes de paquetes muy dispersos que sean difíciles de aislar al instalar varias versiones o arquitecturas del paquete. Use objetos reubicables colectivamente siempre que sea posible.

Nombres de rutas paramétricos

Un *nombre de ruta paramétrico* es un nombre de ruta que contiene la especificación de una variable. Por ejemplo, `/opt/$PKGINST/`*nombre_archivo* es un nombre de ruta paramétrico debido a la especificación de la variable `$PKGINST`. Se *debe* definir un valor predeterminado para especificar la variable en el archivo `pkginfo`. Posteriormente se puede cambiar el valor mediante una secuencia de comandos `request` o `checkinstall`.

La especificación de una variable en una ruta debe comenzar o terminar el nombre de la ruta, o bien estar marcado por barras oblicuas (/). Los nombres paramétricos válidos de rutas adoptan el formato siguiente:

```
$PARAM/tests
tests/$PARAM/generic
/tests/$PARAM
```

La especificación de la variable, una vez definida, puede provocar que la ruta se evalúe como absoluta o reubicable. En el ejemplo siguiente, el archivo `prototype` contiene esta entrada:

```
f none $DIRLOC/tests/generic
```

El archivo `pkginfo` contiene esta entrada:

```
DIRLOC=/myopt
```

El nombre de ruta `$DIRLOC/tests/generic` evalúa al nombre de ruta absoluta `/myopt/tests/generic`, independientemente de si el parámetro `BASEDIR` está configurado en el archivo `pkginfo`.

En este ejemplo, el archivo `prototype` es idéntico al del ejemplo anterior y el archivo `pkginfo` contiene las entradas siguientes:

```
DIRLOC=firstcut
BASEDIR=/opt
```

El nombre de ruta `$DIRLOC/tests/generic` evaluará al nombre de ruta reubicable `/opt/firstcut/tests/generic`.

Para obtener más información sobre nombres de rutas paramétricos, consulte "Uso de directorios base paramétricos" en la página 134.

Unas palabras sobre las ubicaciones de origen y de destino del objeto

El campo *ruta* del archivo `prototype` define dónde se ubicará el objeto en el sistema de destino. Especifique la ubicación actual de los objetos del paquete en el archivo `prototype` si su estructura de directorios no refleja la estructura que se desea en el sistema de destino. Consulte "Organización del contenido de un paquete" en la página 30 para obtener más información sobre la estructuración de objetos en un paquete.

Si el área de desarrollo no está estructurada del mismo modo que desea para el paquete, puede usar el formato *path1=path2* en el campo *ruta*. En este formato, *path1* es la ubicación que el objeto debe tener en el sistema de destino y *path2* es la ubicación que el objeto tiene en su sistema.

También puede usar el formato de nombre de ruta *path1=path2* con *path1* como nombre de objeto reubicable y *path2* como nombre de ruta completo al objeto del sistema.

Nota – Es posible que *path1* no contenga variables de construcción sin definir, sino que contenga variables de instalación sin definir. Es posible que *path2* no contenga variables sin definir, aunque se pueden usar las variables de construcción y de instalación. Para obtener más información sobre la diferencia entre las variables de instalación y de construcción, consulte "Variables de entorno de paquetes" en la página 24.

Los vínculos deben usar el formato *path1= path2* porque los crea el comando `pkgadd`. Como norma general, la *path2* de un vínculo no debe ser nunca absoluta, sino que debe ser relativa a la parte del directorio de *path1*.

Una alternativa al uso del formato *path1=path2* es usar el comando `!search`. Para obtener más información, consulte "Cómo proporcionar una ruta de búsqueda para el comando `pkgmk`" en la página 42.

El campo *mode*

El campo *mode* puede contener un número octal, un signo de interrogación (`?`) o una especificación de variable. Un número octal especifica el modo del objeto cuando se instala en el sistema de destino. Un signo `?` significa que no se cambiará el modo cuando se instale el objeto, lo cual implica que el objeto del mismo nombre ya existe en el sistema de destino.

Una especificación de variable con el formato `$`*mode*, donde la primera letra de la variable debe estar en minúscula, significa que este campo se configurará cuando se construya el paquete. Tenga en cuenta que esta variable se debe definir en el tiempo de la construcción en el archivo `prototype` o como opción del comando `pkgmk`. Para obtener más información sobre la diferencia entre las variables de instalación y de construcción, consulte "Variables de entorno de paquetes" en la página 24.

Los archivos de los tipos `i` (archivo de información), `l` (vínculo físico) y `s` (vínculo simbólico) deben dejar este campo en blanco.

El campo *owner*

El campo *owner* puede contener un nombre de usuario, un signo de interrogación (`?`) o una especificación de variable. Un nombre de usuario tiene un máximo de 14 caracteres y debe ser un nombre que ya exista en el sistema de destino (como `bin` o `root`). Un signo `?` significa que no se cambiará el propietario cuando se instale el objeto, lo cual implica que el objeto del mismo nombre ya existe en el sistema de destino.

Una especificación de variable puede tener el formato $*Owner* o $*propietario*, donde la primera letra de la variable está en mayúsculas o minúsculas. Si la variable comienza por minúsculas, se debe definir cuando se ha construido el paquete, en el archivo `prototype` o como opción para el comando `pkgmk`. Si la variable comienza por una mayúscula, la especificación de la variable se insertará en el archivo `pkginfo` como valor predeterminado, y puede volver a definirse en el tiempo de la instalación mediante una secuencia de comandos `request`. Para obtener más información sobre la diferencia entre las variables de instalación y de construcción, consulte "Variables de entorno de paquetes" en la página 24.

Los archivos de los tipos `i` (archivo de información) y `lb` (vínculo físico) deben dejar este campo en blanco.

El campo *group*

El campo *group* puede contener un nombre de grupo, un signo de interrogación (`?`) o una especificación de variable. Un nombre de grupo tiene un máximo de 14 caracteres y debe ser un nombre que ya exista en el sistema de destino (como `bin` o `sys`). Un signo `?` significa que no se cambiará el grupo cuando se instale el objeto, lo cual implica que el objeto del mismo nombre ya existe en el sistema de destino.

Una especificación de variable puede tener el formato $ *Group* o $*group*, donde la primera letra de la variable está en mayúsculas o minúsculas. Si la variable comienza por minúsculas, se debe definir cuando se ha construido el paquete, en el archivo `prototype` o como opción para el comando `pkgmk`. Si la variable comienza por una mayúscula, la especificación de la variable se insertará en el archivo `pkginfo` como valor predeterminado, y puede volver a definirse en el tiempo de la instalación mediante una secuencia de comandos `request`. Para obtener más información sobre la diferencia entre las variables de instalación y de construcción, consulte "Variables de entorno de paquetes" en la página 24.

Los archivos de los tipos `i` (archivo de información) y `l` (vínculo físico) deben dejar este campo en blanco.

Creación de un archivo `prototype` desde cero

Si desea crear un archivo `prototype` desde cero, puede hacerlo con su editor de textos preferido, y agregarle una entrada por objeto de paquete. Consulte "Formato del archivo `prototype`" en la página 32 y la página de comando man `prototype(4)` para obtener más información sobre el formato de este archivo. Sin embargo, después de haber definido cada objeto de paquete, puede que desee incluir algunas de las funciones que se describen en "Agregación de funciones a un archivo `prototype`" en la página 40.

Ejemplo: creación de un archivo prototype con el comando pkgproto

Puede usar el comando pkgproto para construir un archivo prototype básico, siempre que haya organizado la estructura de directorios de paquetes tal como se describe en "Organización del contenido de un paquete" en la página 30. Por ejemplo, mediante la estructura de directorios de muestra y el archivo pkginfo que se han descrito en las secciones anteriores, los comandos para crear el archivo prototype son los siguientes:

```
$ cd /home/jane
$ pkgproto ./SUNWcadap > InfoFiles/prototype
```

El archivo prototype será, más o menos, así:

```
d none SUNWcadap 0755 jane staff
d none SUNWcadap/demo 0755 jane staff
f none SUNWcadap/demo/file1 0555 jane staff
d none SUNWcadap/srcfiles 0755 jane staff
f none SUNWcadap/srcfiles/file5 0555 jane staff
f none SUNWcadap/srcfiles/file6 0555 jane staff
d none SUNWcadap/lib 0755 jane staff
f none SUNWcadap/lib/file2 0644 jane staff
d none SUNWcadap/man 0755 jane staff
f none SUNWcadap/man/windex 0644 jane staff
d none SUNWcadap/man/man1 0755 jane staff
f none SUNWcadap/man/man1/file4.1 0444 jane staff
f none SUNWcadap/man/man1/file3.1 0444 jane staff
```

Nota – El grupo y el propietario reales de la persona que construye el paquete se registran mediante el comando pkgproto. Una buena técnica es usar los comandos chown -R y chgrp -R, para configurar el propietario y el grupo como se pretendía *antes de* ejecutar el comando pkgproto.

Este archivo prototype de ejemplo no está completo. Consulte la sección siguiente para obtener información sobre la finalización de este archivo.

Ajuste de un archivo prototype creado con el comando pkgproto

Aunque el comando pkgproto es útil para crear un archivo prototype inicial, no crea entradas para cada objeto de paquete que se deba definir. Este comando no hace entradas completas. El comando pkgproto no hace ninguna de las acciones siguientes:

- Crear entradas completas para objetos con tipos de archivos v (archivos volátiles), e (archivos modificables), x (directorios exclusivos) o i (archivos de información o secuencias de comandos de instalación)

- Admitir varias clases con una única invocación

Creación de entradas de objetos con tipos de archivos v, e, x y i

Como mínimo, es necesario modificar el archivo prototype para agregar objetos con el tipo de archivo i. Si ha guardado los archivos de información y las secuencias de comandos de instalación en el primer nivel del directorio de paquetes (por ejemplo /home/jane/SUNWcadap/pkginfo), una entrada del archivo prototype tendría el aspecto siguiente:

```
i pkginfo
```

Si no almacenó los archivos de información ni las secuencias de comandos de instalación en el primer nivel del directorio de paquetes, debe especificar la ubicación de origen. Por ejemplo:

```
i pkginfo=/home/jane/InfoFiles/pkginfo
```

O bien, puede usar el comando !search para especificar la ubicación del comando pkgmk que se debe comprobar al construir el paquete. Consulte "Cómo proporcionar una ruta de búsqueda para el comando pkgmk" en la página 42 para obtener más información.

Para agregar entradas a objetos con los tipos de archivos v, e y x, siga el formato que se describe en "Formato del archivo prototype" en la página 32, o bien consulte la página de comando man prototype(4).

Nota – Recuerde asignar siempre una clase a los archivos con un tipo de archivo e (modificable) y tenga una secuencia de comandos de acción de clase asociada a esa clase. De lo contrario, los archivos se suprimirán durante la eliminación de los paquetes, aunque el nombre de la ruta se guarde con otros paquetes.

Uso de varias definiciones de clases

Si usa el comando pkgproto para crear el archivo prototype básico, puede asignar todos los objetos de paquetes a la clase none o a una clase específica. Tal como se muestra en "Ejemplo: creación de un archivo prototype con el comando pkgproto" en la página 38, el comando pkgproto básico asigna todos los objetos a la clase none. Para asignar todos los objetos a una clase específica, puede usar la opción -c. Por ejemplo:

```
$ pkgproto -c classname /home/jane/SUNWcadap > /home/jane/InfoFiles/prototype
```

Si usa varias clases, puede que necesite modificar manualmente el archivo prototype y modificar el campo *class* para cada objeto. Si usa clases, también debe definir el parámetro CLASSES en el archivo pkginfo y escribir las secuencias de comandos de acción de clase. El uso de clases es una característica optativa que se trata en detalle en "Escritura de secuencias de comandos de acción de clase" en la página 71.

Ejemplo: ajuste de un archivo prototype creado mediante el comando pkgproto

Dado el archivo prototype creado por el comando pkgproto en "Ejemplo: creación de un archivo prototype con el comando pkgproto" en la página 38, es necesario llevar a cabo varias modificaciones.

- Debe haber una entrada para el archivo pkginfo.
- Es necesario cambiar los campos *path* al formato *path1=path2* porque el origen del paquete se encuentra en /home/jane. Debido a que el origen del paquete es un directorio jerárquico y el comando !search no busca de forma recursiva, puede que sea más sencillo usar el formato *path1=path2*.
- Los campos *owner* y *group* deben contener los nombres de usuarios y grupos del sistema de destino. Es decir, la propietaria jane provocará un error porque esta propietaria no forma parte del sistema operativo SunOS.

  El archivo prototype modificado tiene el aspecto siguiente:

```
i pkginfo=/home/jane/InfoFiles/pkginfo
d none SUNWcadap=/home/jane/SUNWcadap 0755 root sys
d none SUNWcadap/demo=/home/jane/SUNWcadap/demo 0755 root bin
f none SUNWcadap/demo/file1=/home/jane/SUNWcadap/demo/file1 0555 root bin
d none SUNWcadap/srcfiles=/home/jane/SUNWcadap/srcfiles 0755 root bin
f none SUNWcadap/srcfiles/file5=/home/jane/SUNWcadap/srcfiles/file5 0555 root bin
f none SUNWcadap/srcfiles/file6=/home/jane/SUNWcadap/srcfiles/file6 0555 root bin
d none SUNWcadap/lib=/home/jane/SUNWcadap/lib 0755 root bin
f none SUNWcadap/lib/file2=/home/jane/SUNWcadap/lib/file2 0644 root bin
d none SUNWcadap/man=/home/jane/SUNWcadap/man 0755 bin bin
f none SUNWcadap/man/windex=/home/jane/SUNWcadap/man/windex 0644 root other
d none SUNWcadap/man/man1=/home/jane/SUNWcadap/man/man1 0755 bin bin
f none SUNWcadap/man/man1/file4.1=/home/jane/SUNWcadap/man/man1/file4.1 0444 bin bin
f none SUNWcadap/man/man1/file3.1=/home/jane/SUNWcadap/man/man1/file3.1 0444 bin bin
```

Agregación de funciones a un archivo prototype

Además de definir cada objeto de paquete en el archivo prototype, también puede hacer lo siguiente:

- Definir objetos adicionales que se deben crear en el tiempo de la instalación.
- Crear vínculos en el tiempo de la instalación.
- Distribuir paquetes en varios volúmenes.
- Anidar archivos prototype.
- Establecer un valor predeterminado para los campos *mode*, *owner* y *group*.
- Proporcionar una ruta de búsqueda para el comando pkgmk.
- Configurar variables de entorno.

Consulte las secciones siguientes para obtener información sobre cómo efectuar estos cambios.

Definición de objetos adicionales que se deben crear en el tiempo de la instalación

Puede usar el archivo `prototype` para definir objetos que no se ofrezcan en el medio de instalación. Durante la instalación, mediante el comando `pkgadd`, estos objetos se crean con los tipos de archivos necesarios, si no existen ya en el tiempo de la instalación.

Para especificar la creación de un objeto en el sistema de destino, agréguele una entrada en el archivo `prototype` con el tipo de archivo adecuado.

Por ejemplo, si desea que se cree un directorio en el sistema de destino, pero no desea ofrecerlo en el medio de instalación, agregue la entrada siguiente para el directorio en el archivo `prototype`:

```
d none /directory 0644 root other
```

Si desea crear un archivo vacío en el sistema de destino, una entrada para el archivo en el archivo `prototype` podría tener el aspecto siguiente:

```
f none filename=/dev/null 0644 bin bin
```

Los únicos objetos que *deben* entregarse en el medio de instalación son archivos regulares y secuencias de comandos de edición (tipos de archivos `e`, `v`, `f`) y los directorios necesarios para incluirlos. Los objetos adicionales se crean sin hacer referencia a los vínculos simbólicos, vínculos físicos, dispositivos, conducciones con nombre, directorios y objetos entregados.

Creación de vínculos en el tiempo de la instalación

Parar crear vínculos durante la instalación de los paquetes, defina lo siguiente en la entrada del archivo `prototype` para el objeto vinculado:

- Su tipo de archivo como `l` (un vínculo) o `s` (un vínculo simbólico).
- El nombre de ruta del objeto vinculado con el formato *path1=path2*, donde *path1* es el destino y *path2* es el archivo de origen. Como norma general, la *path2* de un vínculo no debe ser nunca absoluta, sino que debe ser relativa a la parte del directorio de *path1*. Por ejemplo, un archivo `prototype` que define un vínculo simbólico podría tener el aspecto siguiente:

  ```
  s none etc/mount=../usr/etc/mount
  ```

Los vínculos relativos se especificarían de este modo si el paquete está instalado como reubicable o absoluto.

Distribución de paquetes en varios volúmenes

Cuando construye el paquete con el comando `pkgmk`, éste lleva a cabo los cálculos y las acciones necesarias para organizar un paquete de varios volúmenes. Un paquete de varios volúmenes recibe el nombre de *paquete segmentado*.

Sin embargo, puede usar el campo *part* optativo en el archivo prototype para definir en qué parte desea ubicar un objeto. Un número de este campo anula el comando pkgmk y fuerza la colocación del componente en la parte concreta del campo. Observe que hay una correspondencia de igual a igual entre las partes y los volúmenes para los medios extraíbles formateados como sistemas de archivos. Si el desarrollador asigna previamente los volúmenes, el comando pkgmk emite un error si no hay suficiente espacio en un volumen.

Anidado de archivos prototype

Puede crear varios archivos prototype y después incluirlos mediante el comando !include en el archivo prototype. Anidar archivos puede ser una opción para facilitar el mantenimiento.

En el ejemplo siguiente hay tres archivos prototype. Se está modificando el archivo principal (prototype). Los otros dos archivos (proto2 y proto3) se están incluyendo.

```
!include /source-dir/proto2
!include /source-dir/proto3
```

Configuración de valores predeterminados para los campos *mode*, *owner* y *group*

Con el fin de configurar valores predeterminados para los campos *mode*, *owner* y *group* para objetos de paquetes específicos, puede insertar el comando !default en el archivo prototype. Por ejemplo:

```
!default 0644 root other
```

Nota – El intervalo del comando !default comienza donde se insertó y se extiende hasta el fin del archivo. El intervalo del comando no abarca los archivos incluidos.

Sin embargo, en el caso de directorios (tipo de archivo d) y archivos modificables (tipo de archivo e) que sabe que existe en los sistemas de destino (como /usr o /etc/vfstab), compruebe que los campos *mode*, *owner* y *group* del archivo prototype se establecen como signos de interrogación (?). De este modo no destruirá los valores que un administrador del sitio puede haber modificado.

Cómo proporcionar una ruta de búsqueda para el comando pkgmk

Si la ubicación de origen de los objetos del paquete es diferente de su ubicación de destino y no desea usar el formato *path1=path2* tal como se describe en "Unas palabras sobre las ubicaciones de origen y de destino del objeto" en la página 35, entonces puede usar el comando !search en el archivo prototype.

Por ejemplo, si ha creado un directorio, pkgfiles, en el directorio de inicio, y contiene todos los archivos de información y las secuencias de comandos de instalación, puede determinar que se busque en este directorio cuando el paquete se construya con el comando pkgmk.

El comando del archivo prototype tendría este aspecto:

```
!search /home-dir/pkgfiles
```

Nota – Las solicitudes de búsqueda no abarcan los archivos incluidos. Además, la búsqueda se limita a los directorios específicos enumerados y no se lleva a cabo de forma recursiva.

Configuración de variables de entorno

También puede agregar comandos al archivo `prototype` con el formato `!`*PARAM=value*. Los comandos con este formato definen variables en el entorno actual. Si tiene varios archivos `prototype`, el ámbito de este comando es local para el archivo `prototype` donde se ha definido.

La variable *PARAM* puede comenzar por una letra en minúscula o en mayúscula. Si el valor de la variable *PARAM* no se conoce en el tiempo de la construcción, el comando `pkgmk` se cancela de forma incorrecta. Para obtener más información sobre la diferencia entre las variables de instalación y de construcción, consulte "Variables de entorno de paquetes" en la página 24.

▼ Cómo crear un archivo `prototype` mediante el comando `pkgproto`

Nota – Es más sencillo crear archivos de información y secuencias de comandos de instalación antes de crear un archivo `prototype`. Sin embargo, este orden no es obligatorio. Siempre puede editar el archivo `prototype` después de cambiar el contenido del paquete. Para obtener más detalles sobre los archivos de información y las secuencias de comandos de instalación, consulte Capítulo 3, "Mejora de las funciones de un paquete (tareas)".

1 **Determine qué objetos del paquete serán absolutos y cuáles serán reubicables, si no lo ha hecho ya.**

Para obtener información que le ayude a completar este paso, consulte "El campo *path*" en la página 33.

2 **Organice los objetos del paquete para que reflejen su ubicación en el sistema de destino.**

Si ya ha organizado los paquetes tal como se describe en "Organización del contenido de un paquete" en la página 30, tenga en cuenta que posiblemente necesite hacer algunos cambios basados en sus decisiones en el Paso 1. Si no ha organizado aún el paquete, debería hacerlo ahora. Si no organiza el paquete, no puede usar el comando `pkgproto` para crear un archivo `prototype` básico.

3 **Si el paquete tiene objetos reubicables colectivamente, modifique el archivo pkginfo para configurar el parámetro BASEDIR con el valor adecuado.**

Por ejemplo:

```
BASEDIR=/opt
```

Para obtener información sobre objetos reubicables colectivamente, consulte "Objetos reubicables colectivamente" en la página 34.

4 **Si el paquete tiene objetos reubicables individualmente, cree una secuencia de comandos request para solicitar al instalador el nombre de ruta adecuado. Si lo desea, también puede crear una secuencia de comandos checkinstall para determinar la ruta adecuada desde los datos del sistema de archivos.**

La lista siguiente ofrece números de páginas para su referencia respecto a tareas comunes:

- Para crear una secuencia de comandos request, consulte "Cómo escribir una secuencia de comandos request" en la página 65.
- Para crear una secuencia de comandos checkinstall, consulte "Cómo reunir datos del sistema de archivos" en la página 68.
- Para obtener más información sobre objetos reubicables individualmente, consulte "Objetos reubicables individualmente" en la página 34.

5 **Cambie el propietario y el grupo de todos los componentes del paquete destinados a ser el propietario y el grupo en los sistemas de destino.**

Use los comandos chown -R y chgrp -R en el directorio de paquetes y el directorio de archivos de información.

6 **Ejecute el comando pkgproto para crear un archivo prototype básico.**

El comando pkgproto escanea los directorios para crear un archivo básico. Por ejemplo:

```
$ cd package-directory
$ pkgproto ./package-directory > prototype
```

El archivo prototype se puede ubicar en cualquier lugar del sistema. El mantenimiento de los archivos de información y las secuencias de comandos de instalación en un lugar simplifica el acceso y el mantenimiento. Para obtener información adicional sobre el comando pkgproto, consulte la página de comando man pkgproto(1).

7 **Modifique el archivo prototype con su editor de textos preferido y agregue entradas para los tipos de archivo v, e, x y i.**

Para obtener información sobre los cambios específicos que deba llevar a cabo, consulte "Ajuste de un archivo prototype creado con el comando pkgproto" en la página 38.

8 **(Optativo) Si usa varias clases, modifique los archivos `prototype` y `pkginfo`. Use su editor de textos preferido para hacer los cambios necesarios y cree las secuencias de comandos de clase correspondientes.**

Para obtener información sobre los cambios específicos que puede necesitar, consulte "Ajuste de un archivo `prototype` creado con el comando `pkgproto`" en la página 38 y "Escritura de secuencias de comandos de acción de clase" en la página 71.

9 **Modifique el archivo `prototype` mediante el editor de textos preferido para redefinir los nombres de ruta y cambiar otros valores del campo.**

Para obtener más información, consulte "Ajuste de un archivo `prototype` creado con el comando `pkgproto`" en la página 38.

10 **(Optativo) Modifique el archivo `prototype` mediante el editor de textos preferido para agregar funciones al archivo `prototype`.**

Para obtener más información, consulte "Agregación de funciones a un archivo `prototype`" en la página 40.

11 **Guarde los cambios y salga del editor.**

Véase también Si está listo para la tarea siguiente, consulte "Cómo construir un paquete" en la página 46.

Creación de un paquete

Use el comando `pkgmk` para construir el paquete. El comando `pkgmk` ejecuta las tareas siguientes:

- Coloca todos los objetos definidos del archivo `prototype` en formato del directorio.
- Crea el archivo `pkgmap` que sustituye al archivo `prototype`.
- Produce un paquete instalable que se usa como entrada al comando `pkgadd`.

Uso del comando `pkgmk` más sencillo

La forma más sencilla de este comando es `pkgmk` sin opción alguna. Antes de usar el comando `pkgmk` sin opciones, compruebe que el directorio de trabajo actual contenga el archivo `prototype` del paquete. La salida del comando, archivos y directorios, se escribe en el directorio `/var/spool/pkg`.

El archivo `pkgmap`

Si construye un paquete con el comando `pkgmk`, crea un archivo `pkgmap` que sustituye al archivo `prototype`. El archivo `pkgmap` del ejemplo anterior muestra el contenido siguiente:

```
$ more pkgmap
: 1 3170
1 d none SUNWcadap 0755 root sys
1 d none SUNWcadap/demo 0755 root bin
1 f none SUNWcadap/demo/file1 0555 root bin 14868 45617 837527496
1 d none SUNWcadap/lib 0755 root bin
1 f none SUNWcadap/lib/file2 0644 root bin 1551792 62372 837527499
1 d none SUNWcadap/man 0755 bin bin
1 d none SUNWcadap/man/man1 0755 bin bin
1 f none SUNWcadap/man/man1/file3.1 0444 bin bin 3700 42989 837527500
1 f none SUNWcadap/man/man1/file4.1 0444 bin bin 1338 44010 837527499
1 f none SUNWcadap/man/windex 0644 root other 157 13275 837527499
1 d none SUNWcadap/srcfiles 0755 root bin
1 f none SUNWcadap/srcfiles/file5 0555 root bin 12208 20280 837527497
1 f none SUNWcadap/srcfiles/file6 0555 root bin 12256 63236 837527497
1 i pkginfo 140 10941 837531104
$
```

El formato de este archivo es muy similar al del archivo `prototype`. Sin embargo, el archivo `pkgmap` incluye la información siguiente:

- La primera línea indica el número de volúmenes que abarca el paquete, así como el tamaño aproximado que el paquete tendrá cuando se instale.

  Por ejemplo, `: 1 3170` indica que el paquete abarca un volumen y usará aproximadamente 3170 bloques de 512 bytes cuando se instale.
- Hay tres campos adicionales que definen el tamaño, la suma de comprobación y el tiempo de la modificación de cada objeto de paquete.
- Los objetos del paquete aparecen en orden alfabético por clase y nombre de ruta para reducir el tiempo que dura la instalación del paquete.

▼ Cómo construir un paquete

1 Cree un archivo `pkginfo`, si no lo ha hecho ya.

Para conocer las instrucciones detalladas, consulte "Cómo crear un archivo `pkginfo`" en la página 29.

2 Cree un archivo `prototype`, si no lo ha hecho ya.

Para obtener instrucciones detalladas, consulte "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43.

3 Haga los cambios necesarios para que el directorio de trabajo actual sea el mismo que contiene el archivo `prototype` del paquete.

4 Construya su paquete.

```
$ pkgmk [-o] [-a arch] [-b base-src-dir] [-d device]
   [-f filename] [-l limit] [-p pstamp] [-r rootpath]
   [-v version] [PARAM=value] [pkginst]
```

`-o`	Sobrescribe la versión actual del paquete.
`-a` *arch*	Anula la información de arquitectura del archivo `pkginfo`.
`-b` *base-src-dir*	Solicita que *base-src-dir* se agregue al comienzo de nombres de ruta reubicables cuando el comando `pkgmk` busque objetos en el sistema de desarrollo.
`-d` *device*	Especifica que el paquete debe copiarse en *device* que puede ser un nombre de ruta absoluta de directorio, un disquete o un disco extraíble.
`-f` *filename*	Nombra un archivo, *filename*, que se usa como archivo `prototype`. Los nombres predeterminados son `prototype` o `Prototype`.
`-l` *limit*	Especifica el tamaño máximo, en bloques de 512 bytes, del dispositivo de salida.
`-p` *pstamp*	Anula la definición de indicación de producción en el archivo `pkginfo`.
`-r` *rootpath*	Solicita que el directorio raíz *rootpath* se use para encontrar los objetos en el sistema de desarrollo.
`-v` *version*	Anula la información de la versión del archivo `pkginfo`.
PARAM=value	Configura las variables de entorno globales. Las variables que comienzan por minúscula se resuelven en el tiempo de la construcción. Las que comienzan por mayúscula se sitúan en el archivo `pkginfo` para usarlo en el tiempo de la instalación.
pkginst	Especifica un paquete por su abreviatura o instancia específica (por ejemplo, `SUNWcadap.4`).

Para obtener más información, consulte la página de comando man `pkgmk(1)`.

5 Compruebe el contenido del paquete.

```
$ pkgchk -d device-name pkg-abbrev
Checking uninstalled directory format package pkg-abbrev
from device-name
## Checking control scripts.
## Checking package objects.
## Checking is complete.
$
```

`-d` *device-name*	Especifica la ubicación del paquete. Tenga en cuenta que *device-name* puede ser un nombre de ruta de directorio completo o los identificadores de una cinta o disco extraíble.
pkg-abbrev	Es el nombre de uno o más paquetes (separados por espacios) que se deben comprobar. Si se omite, el comando `pkgchk` comprueba todos los paquetes disponibles.

El comando pkgchk imprime qué aspectos del paquete se comprueban y muestra advertencias o errores, según sea el caso. Para obtener más información sobre el comando pkgchk, consulte "Verificación de la integridad de un paquete" en la página 92.

Precaución – Los errores se deben tener en cuenta especialmente. Un error puede significar que una secuencia de comandos debe corregirse. Compruebe todos los errores y continúe si no está de acuerdo con la salida del comando pkgchk.

Ejemplo 2–2 Creación de un paquete

Este ejemplo usa el archivo prototype creado en "Ajuste de un archivo prototype creado con el comando pkgproto" en la página 38.

```
$ cd /home/jane/InfoFiles
$ pkgmk
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system990708093144"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
$
```

Ejemplo 2–3 Especificación de un directorio de origen para los archivos reubicables

Si el paquete contiene archivos reubicables, puede usar la opción -b *base-src-dir* en el comando pkgmk para especificar un nombre de ruta que agregar al comienzo de los nombres de ruta reubicables mientras el paquete se está creando. Esta opción es útil si no ha usado el formato *path1=path2* para los archivos reubicables ni especificado una ruta de búsqueda con el comando !search en el archivo prototype.

El comando siguiente construye un paquete con las características siguientes:

- El paquete se construye mediante el archivo de ejemplo prototype que se crea mediante el comando pkgproto. Consulte "Ejemplo: creación de un archivo prototype con el comando pkgproto" en la página 38 para obtener más información.

- El paquete se construye sin modificar los campos de *path*.
- El paquete agrega una entrada para el archivo `pkginfo`.

```
$ cd /home/jane/InfoFiles
$ pkgmk -o -b /home/jane
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system960716102636"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
```

En este ejemplo, el paquete se construye en el directorio predeterminado `/var/spool/pkg`, mediante la especificación de la opción `-o`. Esta opción sobrescribe el paquete que se creó en el Ejemplo 2–2.

Ejemplo 2–4 Especificación de diversos directorios de origen para los archivos de información y los objetos de los paquetes

Si coloca archivos de información de paquetes (como `pkginfo` y `prototype`) y objetos de paquetes en dos directorios diferentes, puede crear el paquete si usa las opciones `-b` *base-src-dir* y `-r` *rootpath* en el comando `pkgmk`. Si tiene objetos de paquetes en un directorio llamado `/product/pkgbin` y los demás archivos de información de paquetes en un directorio llamado `/product/pkgsrc`, podría usar el comando siguiente para situar el paquete en el directorio `/var/spool/pkg`:

```
$ pkgmk -b /product/pkgbin -r /product/pkgsrc -f /product/pkgsrc/prototype
```

De forma optativa, puede usar estos comandos para conseguir el mismo resultado:

```
$ cd /product/pkgsrc
$ pkgmk -o -b /product/pkgbin
```

En este ejemplo, el comando `pkgmk` usa el directorio de trabajo actual para buscar las partes restantes del paquete (como los archivos de información `prototype` y `pkginfo`).

Véase también Si desea agregar secuencias de comandos de instalación y archivos de información optativos al paquete, consulte Capítulo 3, "Mejora de las funciones de un paquete (tareas)". De lo contrario, después de construir el paquete, debe verificar su integridad. Capítulo 4, "Verificación y transferencia de un paquete" explica cómo hacerlo y ofrece instrucciones detalladas sobre cómo transferir el paquete verificado a un medio de distribución.

◆ ◆ ◆ CAPÍTULO 3

Mejora de las funciones de un paquete (tareas)

En este capítulo se describe cómo crear secuencias de comandos de instalación y archivos de información optativos para un paquete. Mientras en el Capítulo 2, "Creación de un paquete" se trataban los requisitos mínimos para componer un paquete, este capítulo trata las funciones adicionales que puede realizar en un paquete. Estas funciones adicionales se basan en los criterios que haya considerado al planificar cómo diseñar el paquete. Para obtener más información, consulte "Consideraciones antes de construir un paquete" en la página 17.

A continuación, se muestra una lista de la información general de este capítulo:

Creación de secuencias de comandos de instalación y archivos de información (mapa de tareas)

El mapa de tareas siguiente describe las funciones optativas que puede realizar en un paquete.

TABLA 3–1 Creación de secuencias de comandos de instalación y archivos de información (mapa de tareas)

Tarea	Descripción	Para obtener instrucciones
1. Crear archivos de información	*Definir dependencias de los paquetes* La definición de las dependencias de los paquetes permiten especificar si el paquete es compatible con versiones anteriores, si depende de otros paquetes, o bien si otros paquetes dependen del suyo.	"Cómo definir las dependencias de los paquetes" en la página 53

TABLA 3–1 Creación de secuencias de comandos de instalación y archivos de información (mapa de tareas) *(Continuación)*

Tarea	Descripción	Para obtener instrucciones
	Escribir un mensaje de copyright Un archivo de `copyright` ofrece protección legal a su aplicación de software.	"Cómo escribir un mensaje de copyright" en la página 56
	Reservar espacio adicional en el sistema de destino. Un archivo `space` reserva bloques en el sistema de destino, lo cual le permite crear archivos durante la instalación que no están definidos en el archivo `pkgmap`.	"Cómo reservar espacio adicional en un sistema de destino." en la página 57
2. Crear secuencias de comandos de instalación	*Obtener información del instalador* Una secuencia de comandos `request` permite obtener información de la persona que está instalando su paquete.	"Cómo escribir una secuencia de comandos `request`" en la página 65
	Reunir los datos del sistema de archivos para la instalación Una secuencia de comandos `checkinstall` permite realizar un análisis del sistema de destino y configurar el entorno correcto para la instalación, o bien detenerla adecuadamente.	"Cómo reunir datos del sistema de archivos" en la página 68
	Escribir secuencias de comandos de procedimientos Las secuencias de comandos de procedimientos le permiten ofrecer instrucciones personalizadas de instalación durante fases concretas del proceso de instalación o supresión.	"Cómo escribir secuencias de comandos de procedimientos" en la página 70
	Escribir secuencias de comandos de acción de clase Las secuencias de comandos de acción de clase permiten especificar un conjunto de instrucciones que deben ejecutarse durante la supresión y la instalación de paquetes en grupos específicos de objetos de paquetes.	"Cómo escribir secuencias de comandos de acción de clase" en la página 79

Creación de archivos de información

En esta sección se tratan los archivos de información de paquetes optativos. Gracias a estos archivos puede definir dependencias de paquetes, ofrecer un mensaje de copyright y reservar espacio adicional en un sistema de destino.

Definición de dependencias de paquetes

Necesita determinar si el paquete tiene dependencias respecto a otros paquetes y si otros paquetes dependen del suyo. Es posible definir incompatibilidades y dependencias de paquetes con dos de los archivos de información de paquetes optativos, `compver` y `depend`.

La entrega de un archivo `compver` permite asignar nombres a versiones anteriores del paquete que sean compatibles con el paquete que se está instalando.

La entrega de un archivo `depend` permite definir tres tipos de dependencias asociadas al paquete. Estos tipos de dependencias son las siguientes:

- Un *paquete de prerrequisitos*: su paquete depende de la existencia de otro paquete
- Una *dependencia inversa*: otro paquete depende de la existencia de su paquete

---

Nota – Use el tipo de dependencia inversa sólo si un paquete que no puede entregar un archivo `depend` depende de su paquete.

- Un *paquete incompatible*: su paquete es incompatible con el paquete nombrado

El archivo `depend` soluciona solamente dependencias muy básicas. Si su paquete depende de un archivo concreto, su contenido o su comportamiento, el archivo `depend` no ofrece la precisión adecuada. En este caso se debe usar una secuencia de comandos `request` o `checkinstall` para la comprobación detallada de dependencias. La secuencia de comandos `checkinstall` también es la única capaz de detener correctamente el proceso de instalación de los paquetes.

Nota – Compruebe que los archivos `depend` y `compver` tengan entradas en el archivo `prototype`. El tipo de archivo debe ser `i` (para el archivo de información de paquetes).

Consulte las páginas de comando man `depend(4)` y `compver(4)` para obtener más información.

▼ Cómo definir las dependencias de los paquetes

1. **Convierta el directorio que contenga sus archivos de información en el directorio actual de trabajo.**

2. **Si hay versiones anteriores del paquete y necesita especificar que el nuevo paquete es compatible con ellas, cree un archivo llamado `compver` con su editor de textos preferido.**

   Enumere las versiones con las que su paquete sea compatible. Use este formato:

   string string . . .

El valor de *string* es idéntico al valor asignado al parámetro `VERSION` en el archivo `pkginfo`, para cada paquete compatible.

3 Guarde los cambios y salga del editor.

4 Si su paquete depende de la existencia de otros paquetes, si otros paquetes dependen de la existencia de su paquete, o bien si su paquete es incompatible con otro paquete, cree un archivo llamado `depend` con su editor de textos preferido.

Agregue una entrada para cada dependencia. Use este formato:

type pkg-abbrev pkg-name
(*arch*) *version*
(*arch*) *version* . . .

type — Define el tipo de dependencia. Debe ser uno de los caracteres siguientes: `P` (paquete de prerrequisito), `I` (paquete incompatible), o bien `R` (dependencia inversa).

pkg-abbrev — Especifica la abreviatura del paquete, como `SUNWcadap`.

pkg-name — Especifica el nombre completo del paquete, como `Chip designers need CAD application software to design abc chips. Runs only on xyz hardware and is installed in the usr partition.`

(*arch*) — Opcional. Especifica el tipo de hardware en el que se ejecuta el paquete. Por ejemplo, `sparc` o `x86`. Si especifica una arquitectura, debe usar paréntesis como delimitadores.

version — Opcional. Especifica el valor asignado al parámetro `VERSION` en el archivo `pkginfo`.

Para obtener más información, consulte `depend(4)`.

5 Guarde los cambios y salga del editor.

6 Complete *una* de las tareas siguientes:

- Si desea crear secuencias de comandos de instalación y archivos de información adicionales, vaya a la tarea siguiente, "Cómo escribir un mensaje de copyright" en la página 56.
- Si no ha creado su archivo `prototype`, complete el procedimiento "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43. Vaya al Paso 7.
- Si ya ha creado su archivo `prototype`, edítelo y agregue una entrada para cada archivo que acabe de crear.

7 Construya su paquete.

Consulte "Cómo construir un paquete" en la página 46 si fuera necesario.

Ejemplo 3–1 Archivo `compver`

En este ejemplo, hay cuatro versiones de un paquete: 1.0, 1.1, 2.0 y, el nuevo paquete, 3.0. El nuevo paquete es compatible con las tres versiones anteriores. El archivo `compver` para la versión más reciente podría tener el siguiente aspecto:

```
release 3.0
release 2.0
version 1.1
1.0
```

Las entradas no tienen por qué estar en un orden secuencial. Sin embargo, deben coincidir exactamente con la definición del parámetro `VERSION` en el archivo `pkginfo` de cada paquete. En este ejemplo, los diseñadores de paquetes han utilizado formatos diferentes en las tres primeros versiones.

Ejemplo 3–2 Archivo `depend`

En este ejemplo se da por supuesto que el paquete de ejemplo, `SUNWcadap`, requiere que los paquetes `SUNWcsr` y `SUNWcsu` ya estén instalados en un sistema de destino. El archivo `depend` para `SUNWcadap` tiene el aspecto siguiente:

```
P SUNWcsr Core Solaris, (Root)
P SUNWcsu Core Solaris, (Usr)
```

Véase también Después de construir el paquete, instálelo para confirmar que se instala correctamente y compruebe su integridad. En el Capítulo 4, “Verificación y transferencia de un paquete” se explican estas tareas y se ofrecen instrucciones paso a paso para transferir su paquete verificado a un medio de distribución.

Escritura de un mensaje de copyright

Es necesario decidir si su paquete debe mostrar un mensaje de copyright mientras se está instalando. Si es así, cree el archivo `copyright`.

Nota – Debe incluir un archivo `copyright` para ofrecer protección legal a su aplicación de software. Compruebe con el departamento legal de su empresa las palabras que se deben incluir en el mensaje.

Para entregar un mensaje de copyright, debe crear un archivo llamado `copyright` . Durante la instalación, el mensaje se muestra exactamente tal como aparece en el archivo (sin formato). Consulte la página de comando man `copyright(4)` para obtener más información.

Nota – Asegúrese de que el archivo `copyright` tenga una entrada en el archivo `prototype`. El tipo de archivo debe ser `i` (para el archivo de información de paquetes).

▼ Cómo escribir un mensaje de copyright

1. **Convierta el directorio que contiene sus archivos de información en el directorio de trabajo actual.**

2. **Cree un archivo llamado `copyright` con su editor de textos preferido.**

   Escriba el texto del mensaje de copyright exactamente como desea que aparezca durante la instalación del paquete.

3. **Guarde los cambios y salga del editor.**

4. **Complete *una* de las tareas siguientes:**

   - Si desea crear secuencias de comandos de instalación y archivos de información adicionales, vaya a la tarea siguiente, "Cómo reservar espacio adicional en un sistema de destino." en la página 57.
   - Si *no* ha creado su archivo `prototype`, complete el procedimiento "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43. Vaya al Paso 5.
   - Si ya ha creado su archivo `prototype`, edítelo y agregue una entrada para el archivo de información que acaba de crear.

5. **Construya su paquete.**

   Consulte "Cómo construir un paquete" en la página 46 si fuera necesario.

Ejemplo 3–3 Archivo `copyright`

Por ejemplo, un mensaje de copyright parcial podría tener el aspecto siguiente:

```
Copyright (c) 2003 Company Name
All Rights Reserved

This product is protected by copyright and distributed under
licenses restricting copying, distribution, and decompilation.
```

Véase también Después de construir el paquete, instálelo para confirmar que se instala correctamente y compruebe su integridad. En el Capítulo 4, "Verificación y transferencia de un paquete" se explican estas tareas y se ofrecen instrucciones paso a paso para transferir su paquete verificado a un medio de distribución.

Reserva del espacio adicional en un sistema de destino

Debe determinar si el paquete necesita espacio adicional de disco en el sistema de destino. Este espacio se suma al requerido por los objetos del paquete. Si es así, cree el archivo de información `space`. Esta tarea es diferente de crear archivos y directorios vacíos en el tiempo de la instalación, tal como se puede ver en "Definición de objetos adicionales que se deben crear en el tiempo de la instalación" en la página 41.

El comando `pkgadd` asegura que haya suficiente espacio en el disco para instalar el paquete de acuerdo con las definiciones de objetos del archivo `pkgmap`. Sin embargo, puede que un paquete necesite espacio adicional en el disco más allá de lo que necesiten los objetos definidos en el archivo `pkgmap`. Por ejemplo, puede que el paquete cree un archivo después de la instalación que puede incluir una base de datos, archivos de registro u otro archivo creciente que consuma espacio del disco. Para asegurarse de que haya espacio reservado, debe incluir un archivo `space` que especifique los requisitos de espacio en el disco. El comando `pkgadd` comprueba que exista el espacio adicional especificado en un archivo `space`. Consulte la página de comando man `space(4)` para obtener más información.

Nota – Asegúrese de que el archivo `space` tenga una entrada en el archivo `prototype`. El tipo de archivo debe ser `i` (para el archivo de información de paquetes).

▼ Cómo reservar espacio adicional en un sistema de destino.

1 Convierta el directorio que contiene sus archivos de información en el directorio de trabajo actual.

2 Cree un archivo llamado `space` con su editor de textos preferido.

Especifique los requisitos adicionales para el espacio en el disco que su paquete necesita. Use este formato:

pathname blocks inodes

pathname Especifica un nombre de directorio que puede ser, aunque no necesariamente, el punto de montaje de un sistema de archivos.

blocks Especifica el número de bloques de 512 bytes que desea reservar.

inodes Especifica el número de inodes necesarios.

Para obtener más información, consulte la página de comando man `space(4)`.

3 Guarde los cambios y salga del editor.

4 Complete una de las tareas siguientes.

- Si desea crear secuencias de comandos de instalación, vaya a la tarea siguiente, "Cómo escribir una secuencia de comandos `request`" en la página 65.
- Si no ha creado su archivo `prototype`, complete el procedimiento que se indica en "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43. Vaya al Paso 5.
- Si ya ha creado su archivo `prototype`, edítelo y agregue una entrada para el archivo de información que acaba de crear.

5 Construya su paquete.

Consulte "Cómo construir un paquete" en la página 46 si fuera necesario.

Ejemplo 3–4 Archivo `space`

Este archivo `space` de ejemplo especifica que se reserven 1000 bloques de 512 bytes y 1 inode en el directorio `/opt` del sistema de destino.

```
/opt   1000   1
```

Véase también Después de construir el paquete, instálelo para confirmar que se instala correctamente y compruebe su integridad. En el Capítulo 4, "Verificación y transferencia de un paquete" se explican estas tareas y se ofrecen instrucciones paso a paso para transferir su paquete verificado a un medio de distribución.

Creación de secuencias de comandos de instalación

En esta sección se tratan las secuencias de comandos de instalación de paquetes optativos. El comando `pkgadd` ejecuta automáticamente todas las acciones necesarias para instalar un paquete mediante los archivos de información de paquetes como entrada. *No* es necesario que suministre secuencias de comandos de instalación de paquetes. Sin embargo, si desea crear

procedimientos de instalación personalizados para su paquete, puede hacerlo con secuencias de comandos de instalación. Secuencias de comandos de instalación:

- Debe ser ejecutable por parte del shell Bourne (`sh`)
- Debe contener comandos shell Bourne y texto
- No es necesario que incluya el identificador de shell`#!/bin/sh`
- No es necesario que sea un archivo ejecutable

Hay cuatro tipos de secuencias de comandos de instalación con los que puede llevar a cabo acciones personalizadas:

- La secuencia de comandos `request`

  La secuencia de comandos `request` solicita datos del administrador que instala un paquete para asignar o redefinir variables de entornos.

- La secuencia de comandos `checkinstall`

  La secuencia de comandos `checkinstall` busca en el sistema de destino los datos necesarios, puede establecer o modificar variables de entorno de paquetes y determina si continúa la instalación.

  Nota – La secuencia de comandos `checkinstall` está disponible a partir de Solaris 2.5 y versiones compatibles.

- Secuencias de comandos de procedimientos

  Las secuencias de comandos de procedimientos identifican la invocación de un procedimiento antes o después de la instalación o eliminación de un paquete. Las cuatro secuencias de comandos de procedimientos son `preinstall`, `postinstall`, `preremove` y `postremove`.

- Secuencias de comandos de acción de clase

  Las secuencias de comandos de acción de clase definen una acción o conjunto de acciones que se deben aplicar a una clase de archivos durante la instalación o eliminación. Puede definir sus propias clases. Si lo desea, también puede usar una de las cuatro clases estándar (`sed`, `awk`, `build` y `preserve`).

Proceso de secuencias de comandos durante la instalación de paquetes

El tipo de secuencias de comandos que utilice depende de cuándo se necesite la acción de la secuencia de comandos durante el proceso de instalación. Al instalar un paquete, el comando `pkgadd` ejecuta los pasos siguientes:

1. Ejecuta la secuencia de comandos `request`.

   Este paso es el único punto en que su paquete puede solicitar alguna acción del administrador que esté instalando el paquete.

2. Ejecuta la secuencia de comandos `checkinstall`.

   La secuencia de comandos `checkinstall` reúne datos del sistema de archivos y puede crear o modificar definiciones de variables de entorno para controlar la instalación posterior. Para obtener más información sobre las variables de entorno del paquete, consulte "Variables de entorno de paquetes" en la página 24.

3. Ejecuta la secuencia de comandos `preinstall`.

4. Instala objetos del paquete, para cada clase que se vaya a instalar.

   La instalación de estos archivos se produce clase a clase; las secuencias de comandos de acción de clase se ejecutan consecuentemente. La lista de clases en las que se trabaja y el orden en el que se deben instalar se define inicialmente con el parámetro `CLASSES` en el archivo `pkginfo`. Sin embargo, la secuencia de comandos `request` o `checkinstall` puede cambiar el valor del parámetro `CLASSES`. Para obtener más información sobre cómo se procesan las clases durante la instalación, consulte "Cómo se procesan las clases durante la instalación de los paquetes" en la página 72.

   a. Crea vínculos simbólicos, dispositivos, conducciones con nombre y los directorios necesarios.

   b. Instala los archivos regulares (tipos de archivos `e`, `v`, `f`), de acuerdo con su clase

      Sólo se pasan archivos regulares para instalar a la secuencia de comandos de acción de clase. Todos los demás objetos de paquetes se crean de forma automática a partir de la información del archivo `pkgmap`.

   c. Crea todos los vínculos físicos.

5. Ejecuta la secuencia de comandos `postinstall`.

Proceso de secuencia de comandos durante la eliminación de paquetes

Cuando se suprime un paquete, el comando `pkgrm` ejecuta estos pasos:

1. Ejecuta la secuencia de comandos `preremove`.

2. Suprime los objetos de paquetes para cada clase

   La eliminación también se produce clase a clase. Las secuencias de comandos de eliminación también se procesan en el orden inverso de la instalación, de acuerdo con la secuencia definida en el parámetro `CLASSES`. Para obtener más información sobre cómo se procesan las clases durante la instalación, consulte "Cómo se procesan las clases durante la instalación de los paquetes" en la página 72.

a. Suprime los vínculos físicos.
b. Suprime los archivos regulares.
c. Suprime los vínculos simbólicos, los dispositivos y las conducciones con nombre.

3. Ejecuta la secuencia de comandos `postremove`.

La secuencia de comandos `request` no se procesa en el tiempo de la supresión de paquetes. Sin embargo, la salida de la secuencia de comandos se retiene en el paquete instalado y queda disponible para las secuencias de comandos de supresión. La salida de la secuencia de comandos `request` es una lista de variables de entorno.

Variables de entorno de paquetes disponibles para las secuencias de comandos

Los grupos siguientes de variables de entorno están disponibles para todas las secuencias de comandos de instalación. Algunas de las variables de entorno se pueden modificar mediante una secuencia de comandos `request` o `checkinstall`.

- La secuencia de comandos `request` o `checkinstall` puede establecer o modificar cualquiera de los parámetros estándar del archivo `pkginfo`, excepto en el caso de los parámetros necesarios. Los parámetros estándar de instalación se describen detalladamente en la página de comando man `pkginfo(4)`.

  Nota – El parámetro `BASEDIR` sólo se puede modificar a partir de la versión Solaris 2.5 y versiones compatibles.

- Puede definir sus propias variables de entorno de instalación si les asigna valores en el archivo `pkginfo`. Estas variables de entorno deben ser alfanuméricas con las iniciales en mayúsculas. Cualquiera de las variables de entorno se puede modificar mediante una secuencia de comandos `request` o `checkinstall`.
- Las secuencias de comandos `request` y `checkinstall` pueden definir nuevas variables de entorno si se les asignan valores y se colocan en el entorno de instalación.
- En la tabla siguiente se muestran variables de entorno que están disponibles para todas las secuencias de comandos de instalación a través del entorno. Una secuencia de comandos no puede modificar ninguna de estas variables de entorno.

Variable de entorno	Descripción
CLIENT_BASEDIR	Directorio base respecto al sistema de destino. Mientras BASEDIR es la variable que usar si hace referencia a un objeto de paquete específico desde el sistema de instalación (muy probablemente un servidor), CLIENT_BASEDIR es la ruta que incluir en los archivos situados en el sistema cliente. CLIENT_BASEDIR existe si BASEDIR también existe y es idéntico a BASEDIR si no existe PKG_INSTALL_ROOT.
INST_DATADIR	Directorio donde se encuentra en paquete que se está leyendo. Si se está leyendo el paquete de una cinta, esta variable será la ubicación de un directorio temporal donde el paquete se haya transferido al formato de directorio. En otras palabras: si se supone que no hay una extensión para el nombre del paquete (por ejemplo, SUNWstuff.d), la secuencia de comandos request del paquete actual se encontraría en $INST_DATADIR/$PKG/install.
PATH	Lista de búsqueda utilizada por sh para buscar comandos en la invocación de la secuencia de comandos. PATH se establece normalmente en /sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin.
PKGINST	Identificador de instancias del paquete que se está instalando. Si no hay instalada otra instancia del paquete, el valor será la abreviatura del paquete (por ejemplo, SUNWcadap). De lo contrario, el valor es la abreviatura del paquete seguida de un sufijo, como SUNWcadap.4.
PKGSAV	Directorio donde se pueden guardar los archivos para que los utilicen las secuencias de comandos de eliminación o donde se pueden encontrar archivos guardados anteriormente. Disponible solamente en Solaris 2.5 y versiones compatibles.
PKG_CLIENT_OS	Sistema operativo del cliente donde se está instalando el paquete. El valor de esta variable Solaris.
PKG_CLIENT_VERSION	Versión de Solaris con formato x.y.
PKG_CLIENT_REVISION	Revisión de la generación de Solaris.
PKG_INSTALL_ROOT	El sistema de archivos raíz del sistema de destino donde el paquete se está instalando. Esta variable sólo existe si los comandos pkgadd y pkgrm se han invocado con la opción -R. Esta existencia condicional facilita su uso en las secuencias de comandos de procedimientos con el formato ${PKG_INSTALL_ROOT}/*somepath*.
PKG_NO_UNIFIED	Variable de entorno que se establece si los comandos pkgadd y pkgrm se invocaron con las opciones -M y -R. Esta variable de entorno pasa a algún comando de paquetes o secuencia de comandos de instalación de paquetes que sea parte del entorno de paquetes.
UPDATE	Esta variable de entorno no existe en la mayoría de los entornos de instalación. Si esta variable existe (con el valor yes), puede tener dos significados: o bien ya se ha instalado en el sistema un paquete con el mismo nombre, versión y arquitectura, o bien este paquete sobrescribe uno ya instalado con el mismo nombre en la dirección del administrador. En estos casos, siempre se usa el directorio base original.

Obtención de información del paquete para una secuencia de comandos

Se pueden usar dos comandos de secuencias de comandos para solicitar información sobre un paquete:

- El comando `pkginfo` devuelve información sobre paquetes de software, como el identificador de instancias y el nombre del paquete.
- El comando `pkgparam` devuelve valores para las variables de entorno solicitadas.

  Consulte las páginas de comando man `pkginfo(1)` y `pkgparam(1)`, así como el Capítulo 4, "Verificación y transferencia de un paquete", si desea más información.

Códigos de salida para secuencias de comandos

Cada secuencia de comandos debe salir con uno de los códigos de salida que aparecen en la tabla siguiente.

TABLA 3–2 Instalación de códigos de salida de secuencias de comandos

Código	Significado
0	Finalización satisfactoria de la secuencia de comandos.
1	Error fatal. El proceso de instalación se ha terminado en este momento.
2	Advertencia o condición de posible error. La instalación continúa. Aparece un mensaje de advertencia en el tiempo de la finalización.
3	El comando `pkgadd` se ha detenido de forma correcta. Sólo la secuencia de comandos `checkinstall` devuelve este código.
10	Se debe reiniciar el sistema cuando la instalación de todos los paquetes seleccionados se haya completado. (Este valor se debe agregar a uno de los códigos de salida de un único dígito).
20	El sistema se debe reiniciar inmediatamente al terminar la instalación del paquete actual. (Este valor se debe agregar a uno de los códigos de salida de un único dígito).

Consulte el Capítulo 5, "Casos prácticos de creación de paquetes" si desea ver ejemplos de códigos de salida devueltos por las secuencias de comandos de la instalación.

Nota – Todas las secuencias de comandos de instalación distribuidas con su paquete deben contar con una entrada en el archivo `prototype`. El tipo de archivo debe ser `i` (para la secuencia de comandos de instalación del paquete).

Escritura de una secuencia de comandos `request`

La secuencia de comandos `request` es el único modo de que su paquete pueda interactuar directamente con el administrador que lo instale. Esta secuencia de comandos se puede usar, por ejemplo, para preguntar al administrador si deben instalarse partes optativas de un paquete.

La salida de una secuencia de comandos `request` debe ser una lista de variables de entorno y sus valores. Esta lista puede incluir alguno de los parámetros que haya creado en el archivo `pkginfo`, así como los parámetros `CLASSES` y `BASEDIR`. La lista también puede presentar variables de entorno que no se hayan definido en otro lugar. Sin embargo, el archivo `pkginfo` debe ofrecer siempre valores predeterminados cuando convenga. Para obtener más información sobre las variables de entorno del paquete, consulte "Variables de entorno de paquetes" en la página 24.

Cuando la secuencia de comandos `request` asigna valores a una variable de entorno, debe conseguir que esos valores estén disponibles para el comando `pkgadd` y otras secuencias de comandos de paquetes.

Comportamientos de la secuencia de comandos `request`

- La secuencia de comandos `request` no puede modificar archivos. Esta secuencia de comandos sólo interactúa con administradores que instalen el paquete y crea una lista de asignaciones de variables de entorno basadas en esa interacción. La secuencia de comandos `request` se ejecuta como el usuario sin privilegios `noaccess` si existe dicho usuario. De lo contrario, la secuencia de comandos se ejecuta como `root`.
- El comando `pkgadd` llama a la secuencia de comandos `request` con un argumento que nombra el archivo de respuesta de la secuencia de comandos. El archivo de respuesta almacena las respuestas el administrador.
- La secuencia de comandos `request` no se ejecuta durante la eliminación de paquetes. Sin embargo, las variables de entorno asignadas por la secuencia de comandos se guardan y están disponibles durante la eliminación de paquetes.

Normativa de diseño para secuencias de comandos `request`

- Sólo puede haber una secuencia de comandos `request` por paquete. La secuencia de comandos debe llevar el nombre de `request`.
- Las asignaciones de variables de entorno se deben agregar al entorno de instalación para que las use el comando `pkgadd` y otras secuencias de comandos de creación de paquetes; para ello, escríbalas en el archivo de respuesta (conocido para la secuencia de comandos como `$1`).
- Las variables de entorno del sistema y las variables de entorno de instalación estándar, excepto en el caso de los parámetros `CLASSES` y `BASEDIR`, no las puede modificar una secuencia de comandos `request`. Puede cambiar cualquiera de las demás variables de entorno que haya creado.

Nota – Una secuencia de comandos `request` sólo puede modificar el parámetro `BASEDIR` a partir de Solaris 2.5 y versiones compatibles.

- A cada variable de entorno que la secuencia de comandos `request` pueda manipular se debe asignar un valor predeterminado en el archivo `pkginfo`.
- El formato de la lista de salida debe ser *PARAM=value*. Por ejemplo:

  ```
  CLASSES=none class1
  ```
- El terminal del administrador debe definirse como entrada estándar a la secuencia de comandos `request`.
- No lleve a cabo análisis especiales del sistema de destino en una secuencia de comandos `request`. Es arriesgado buscar en el sistema la presencia de determinados binarios o comportamientos, así como establecer las variables de entorno de acuerdo con ese análisis. No hay garantías de que la secuencia de comandos `request` se ejecutará realmente en el tiempo de la instalación. El administrador que instale el paquete puede proporcionar un archivo de respuesta que insertará las variables de entorno sin llamar nunca a la secuencia de comandos `request`. Si la secuencia de comandos `request` también evalúa el sistema de archivos de destino, puede que dicha evaluación no tenga lugar. Es mejor dejar el análisis del sistema de destino para que reciba un tratamiento especial a la secuencia de comandos `checkinstall`.

Nota – Si los administradores que instalen el paquete pudieran utilizar JumpStart, la instalación del paquete no debe ser interactiva. No debe proporcionar una secuencia de comandos `request` con su paquete, o bien debe comunicar a los administradores que deben usar el comando `pkgask` antes de la instalación. El comando `pkgask` guarda sus respuestas en la secuencia de comandos `request`. Para obtener más información sobre el comando `pkgask`, consulte la página de comando man `pkgask(1M)`.

▼ Cómo escribir una secuencia de comandos `request`

1. **Convierta el directorio que contiene sus archivos de información en el directorio de trabajo actual.**
2. **Cree un archivo llamado `request` con su editor de textos preferido.**
3. **Guarde los cambios y salga del editor cuando haya acabado.**
4. **Complete una de las tareas siguientes.**

   - Si desea crear secuencias de comandos de instalación adicionales, vaya a la tarea siguiente, "Cómo reunir datos del sistema de archivos" en la página 68.

- Si no ha creado su archivo `prototype`, complete el procedimiento "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43. Vaya al Paso 5.
- Si ya ha creado su archivo `prototype`, edítelo y agregue una entrada para la secuencia de comandos de instalación que acabe de crear.

5 Construya su paquete.

Consulte "Cómo construir un paquete" en la página 46 si fuera necesario.

Ejemplo 3–5 Escritura de una secuencia de comandos `request`

Cuando una secuencia de comandos `request` asigna valores a las variables de entorno, debe hacer que esos valores estén disponibles para el comando `pkgadd`. Este ejemplo muestra un segmento de secuencia de comandos `request` que ejecuta esta tarea para las cuatro variables de entorno: `CLASSES`, `NCMPBIN`, `EMACS` y `NCMPMAN`. Suponga que estas variables se definieron en una sesión interactiva con el administrador anteriormente, en la secuencia de comandos.

```
# make environment variables available to installation
# service and any other packaging script we might have

cat >$1 <<!
CLASSES=$CLASSES
NCMPBIN=$NCMPBIN
EMACS=$EMACS
NCMPMAN=$NCMPMAN
!
```

Véase también Después de construir el paquete, instálelo para confirmar que se instala correctamente y compruebe su integridad. En el Capítulo 4, "Verificación y transferencia de un paquete" se explican estas tareas y se ofrecen instrucciones paso a paso para transferir su paquete verificado a un medio de distribución.

Reunión de datos del sistema de archivos con la secuencia de comandos `checkinstall`

La secuencia de comandos `checkinstall` se ejecuta poco después de la secuencia de comandos `request` optativa. La secuencia de comandos `checkinstall` se ejecuta como el usuario `noaccess` si dicho usuario existe. La secuencia de comandos `checkinstall` no tiene autoridad para cambiar los datos del sistema de archivos. Sin embargo, de acuerdo con la información que la secuencia de comandos reúne, puede crear o modificar variables de entorno, con el fin de controlar el curso de la instalación resultante. La secuencia de comandos también es capaz de detener correctamente el proceso de instalación.

La secuencia de comandos `checkinstall` tiene como finalidad ejecutar comprobaciones básicas en un sistema de archivos que no sería normal para el comando `pkgadd`. Por ejemplo,

esta secuencia de comandos se puede usar para determinar si algunos archivos del paquete actual sobrescribirán los archivos existentes, o bien para administrar dependencias de software general. El archivo `depend` sólo administra dependencias de paquetes.

A diferencia de la secuencia de comandos `request`, `checkinstall` se ejecuta independientemente de si se ha proporcionado un archivo de respuesta. La presencia de la secuencia de comandos no marca el paquete como interactivo. La secuencia de comandos `checkinstall` se puede usar en situaciones en las que se ha prohibido una secuencia de comandos `request`, o bien una interacción administrativa no es práctica.

Nota – La secuencia de comandos `checkinstall` está disponible a partir de Solaris 2.5 y versiones compatibles.

Comportamientos de la secuencia de comandos `checkinstall`

- La secuencia de comandos `checkinstall` no puede modificar archivos. Esta secuencia de comandos sólo analiza el estado del sistema y crea una lista de asignaciones de variables de entorno basada en esta interacción. Para reforzar esta restricción, la secuencia de comandos `checkinstall` se ejecuta como el usuario sin privilegios `noaccess` si dicho usuario existe. De lo contrario, la secuencia de comandos se ejecuta como usuario sin privilegios `nobody`. La secuencia de comandos `checkinstall` no tiene autoridad de superusuario.
- El comando `pkgadd` llama a la secuencia de comandos `checkinstall` con un argumento que nombra el archivo de respuesta de la secuencia de comandos. El archivo de respuesta de la secuencia de comandos es el archivo que guarda las respuestas del administrador.
- La secuencia de comandos `checkinstall` no se ejecuta durante la eliminación de paquetes. Sin embargo, las variables de entorno asignadas por la secuencia de comandos se guardan y están disponibles durante la eliminación de paquetes.

Normativa de diseño para secuencias de comandos `checkinstall`

- Sólo puede haber una secuencia de comandos `checkinstall` por paquete. La secuencia de comandos debe llevar el nombre de `checkinstall`.
- Las asignaciones de variables de entorno se deben agregar al entorno de instalación para que las use el comando `pkgadd` y otras secuencias de comandos de creación de paquetes; para ello, escríbalas en el archivo de respuesta (conocido para la secuencia de comandos como `$1`).
- Las variables de entorno del sistema y las variables de entorno de instalación estándar, excepto en el caso de los parámetros `CLASSES` y `BASEDIR`, no las puede modificar una secuencia de comandos `checkinstall`. Puede cambiar cualquiera de las demás variables de entorno que haya creado.
- A cada variable de entorno que la secuencia de comandos `checkinstall` pueda manipular se debe asignar un valor predeterminado en el archivo `pkginfo`.

- El formato de la lista de salida debe ser *PARAM=value*. Por ejemplo:

  ```
  CLASSES=none class1
  ```

- No se permite la interacción con el administrador durante la ejecución de una secuencia de comandos `checkinstall`. Toda la interacción con el administrador se restringe a la secuencia de comandos `request`.

▼ Cómo reunir datos del sistema de archivos

1. **Convierta el directorio que contiene sus archivos de información en el directorio de trabajo actual.**

2. **Cree un archivo llamado `checkinstall` con su editor de textos preferido.**

3. **Guarde los cambios y salga del editor cuando haya acabado.**

4. **Complete una de las tareas siguientes.**

   - Si desea crear secuencias de comandos de instalación adicionales, vaya a la tarea siguiente, "Cómo escribir secuencias de comandos de procedimientos" en la página 70.
   - Si no ha creado su archivo `prototype`, complete el procedimiento "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43. Vaya al Paso 5.
   - Si ya ha creado su archivo `prototype`, edítelo y agregue una entrada para la secuencia de comandos de instalación que acabe de crear.

5. **Construya su paquete.**

   Consulte "Cómo construir un paquete" en la página 46 si fuera necesario.

Ejemplo 3–6 Escritura de una secuencia de comandos `checkinstall`

Esta secuencia de comandos de ejemplo `checkinstall` comprueba si el software de base de datos necesitado por el paquete `SUNWcadap` está instalado.

```
# checkinstall script for SUNWcadap
#
# This confirms the existence of the required specU database

# First find which database package has been installed.
pkginfo -q SUNWspcdA    # try the older one

if [ $? -ne 0 ]; then
   pkginfo -q SUNWspcdB    # now the latest

      if [ $? -ne 0 ]; then    # oops
            echo "No database package can be found. Please install the"
```

```
            echo "SpecU database package and try this installation again."
            exit 3        # Suspend
      else
            DBBASE="`pkgparam SUNWsbcdB BASEDIR`/db"    # new DB software
      fi
else
      DBBASE="`pkgparam SUNWspcdA BASEDIR`/db"    # old DB software
fi

# Now look for the database file we will need for this installation
if [ $DBBASE/specUlatte ]; then
      exit 0        # all OK
else
      echo "No database file can be found. Please create the database"
      echo "using your installed specU software and try this"
      echo "installation again."
      exit 3        # Suspend
fi
```

Véase también Después de construir el paquete, instálelo para confirmar que se instala correctamente y compruebe su integridad. En el Capítulo 4, “Verificación y transferencia de un paquete” se explican estas tareas y se ofrecen instrucciones paso a paso para transferir su paquete verificado a un medio de distribución.

Escritura de secuencias de comandos de procedimientos

Las secuencias de comandos de procedimientos ofrecen un conjunto de instrucciones que llevar a cabo en fases concretas de la eliminación o la instalación de paquetes. Las cuatro secuencias de comandos de procedimientos deben tener uno de los nombres predefinidos, según cuándo se vayan a ejecutar las instrucciones. Las secuencias de comandos se ejecutan sin argumentos.

- La secuencia de comandos `preinstall`

  se ejecuta antes de que comience la instalación de la clase. Esta secuencia de comandos no debe instalar archivos.

- La secuencia de comandos `postinstall`

  se ejecuta después de que se hayan instalado todos los volúmenes.

- La secuencia de comandos `preremove`

  se ejecuta antes de que comience la eliminación de la clase. Esta secuencia de comandos no debe suprimir archivos.

- La secuencia de comandos `postremove`

  se ejecuta después de todas las clases se hayan suprimido.

Comportamientos de secuencias de comandos de procedimientos

Las secuencias de comandos de procedimientos se ejecutan como `uid=root` y `gid=other`.

Normativa de diseño para secuencias de comandos de procedimientos

- Cada secuencia de comandos debe poder ejecutarse más de una vez porque se ejecuta una vez para cada volumen de un paquete. Esto significa que la ejecución de una secuencia de comandos un determinado número de veces con la misma entrada produce los mismos resultados que la ejecución de la secuencia de comandos sólo una vez.
- Cada secuencia de comandos de procedimiento que instale un objeto de paquete que no se encuentre en el archivo `pkgmap` debe usar el comando `installf` para avisar a la base de datos del paquete que agrega o modifica el nombre de una ruta. Cuando se completen todas las adiciones o modificaciones, este comando debe invocarse con la opción `-f`. Sólo las secuencias de comandos `postinstall` y `postremove` pueden instalar objetos de paquetes de este modo. Consulte la página de comando man `installf(1M)` y el Capítulo 5, "Casos prácticos de creación de paquetes", si desea obtener más información.
- No se permite la interacción con el administrador durante la ejecución de una secuencia de comandos de procedimiento. Toda la interacción con el administrador se restringe a la secuencia de comandos `request`.
- Cada secuencia de comandos de procedimientos que suprime archivos no instalados del archivo `pkgmap` debe usar el comando `removef` para avisar a la base de datos del paquete que está suprimiendo un nombre de ruta. Cuando la supresión se haya completado, este comando debe invocarse con la opción `-f`. Consulte la página de comando man `removef(1M)` y el Capítulo 5, "Casos prácticos de creación de paquetes", para obtener más información y ejemplos.

Nota – Los comandos `installf` y `removef` se deben usar porque las secuencias de comandos de procedimientos no se asocian automáticamente con los nombres de rutas que aparecen en el archivo `pkgmap`.

▼ Cómo escribir secuencias de comandos de procedimientos

1 Convierta el directorio que contiene sus archivos de información en el directorio de trabajo actual.

2 Cree una o más secuencias de comandos de procedimientos con su editor de texto preferido.

Una secuencia de comandos de procedimientos debe tener uno de los nombres predefinidos: `preinstall`, `postinstall`, `preremove` o `postremove`.

3 **Guarde los cambios y salga del editor.**

4 **Complete una de las tareas siguientes.**

- Si desea crear secuencias de comandos de acción de clase, vaya a la tarea siguiente, "Cómo escribir secuencias de comandos de acción de clase" en la página 79.
- Si no ha creado su archivo `prototype`, complete el procedimiento "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43. Vaya al Paso 5.
- Si ya ha creado su archivo `prototype`, edítelo y agregue una entrada para cada secuencia de comandos de instalación que acabe de crear.

5 **Construya su paquete.**

Consulte "Cómo construir un paquete" en la página 46 si fuera necesario.

Véase también Después de construir el paquete, instálelo para confirmar que se instala correctamente y compruebe su integridad. En el Capítulo 4, "Verificación y transferencia de un paquete" se explican estas tareas y se ofrecen instrucciones paso a paso para transferir su paquete verificado a un medio de distribución.

Escritura de secuencias de comandos de acción de clase

Definición de clases de objetos

Las clases de objetos permiten una serie de acciones que se deben ejecutar en un grupo de objetos de paquetes en la instalación o la eliminación. Asigne objetos a una clase en el archivo `prototype`. A todos los objetos del paquete se les debe proporcionar una clase, aunque la clase `none` se utiliza de forma predeterminada para objetos que no requieran una acción especial.

El parámetro de instalación `CLASSES`, definido en el archivo `pkginfo`, es una lista de clases que se deben instalar (incluida la clase `none`).

Nota – Los objetos definidos en el archivo `pkgmap` que pertenecen a una clase que no aparece en este parámetro del archivo `pkginfo` *no* se instalarán.

La lista `CLASSES` determina el orden de instalación. La clase `none` siempre se instala en primer lugar, en el caso de que exista, y se suprime en último lugar. Puesto que los directorios son la estructura de apoyo fundamental para los demás objetos del sistema de archivos, deben asignarse todos a la clase `none`. Se pueden hacer excepciones, pero como norma general, la clase `none` es la más segura. Esta estrategia asegura que los directorios se crean antes de los objetos que incluirán. Asimismo, no se efectúa ningún intento de suprimir un directorio antes de que se haya vaciado.

Cómo se procesan las clases durante la instalación de los paquetes

A continuación se describen las acciones del sistema que tienen lugar cuando se instala una clase. Las acciones se repiten una vez para cada volumen de un paquete, durante la instalación de ese volumen.

1. El comando `pkgadd` crea una lista de nombre de ruta.

   El comando `pkgadd` crea una lista de nombres de ruta sobre la que opera la secuencia de comandos de la acción. Cada línea de la lista contiene nombres de ruta de origen y de destino, separados por un espacio. El nombre de ruta de origen indica dónde reside el objeto que se debe instalar en el volumen de instalación. El nombre de ruta de destino indica la ubicación del sistema de destino donde el objeto se debe instalar. El contenido de la lista se restringe mediante los criterios siguientes:

   - La lista sólo contiene nombres de ruta que pertenecen a la clase asociada.
   - Si falla el intento de crear el objeto del paquete, los directorios, las conducciones con nombres, los dispositivos de caracteres, los dispositivos de bloques y los vínculos simbólicos se incluyen en la lista con el nombre de ruta de origen establecido en `/dev/null`. Normalmente, el comando `pkgadd` crea automáticamente estos elementos (en el caso de que no existieran) y se les otorga los atributos pertinentes (modo, propietario, grupo), tal como se define en el archivo `pkgmap`.
   - Los archivos vinculados donde el tipo de archivo es `l` no se incluyen en la lista bajo ninguna circunstancia. Los vínculos físicos de la clase proporcionada se crean en el elemento 4.

2. Si no se proporciona una secuencia de comandos de acción de clase para la instalación de una clase determinada, los nombres de rutas de la lista generada se copian del volumen a la ubicación de destino pertinente.

3. Se ejecutará una secuencia de comandos de acción de clase en el caso de que exista.

   La secuencia de comandos de acción de clase se ejecuta con la entrada estándar que contiene la lista que se genera en el elemento 1. Este volumen es el último, o en esta clase no hay más objetos, la secuencia de comandos se ejecuta con el único argumento de `ENDOFCLASS`.

   Nota – Incluso si no existen archivos regulares de esta clase en el paquete, se llama a la secuencia de comandos de acción de clase al menos una vez con una lista vacía y el argumento `ENDOFCLASS`.

   ---

4. El comando `pkgadd` lleva a cabo una auditoría de atributos y contenido, además de crear vínculos físicos.

   Después de ejecutar correctamente los elementos 2 o 3, el comando `pkgadd` efectúa una auditoría de la información de atributos y contenido para la lista de nombres de ruta. El comando `pkgadd` crea los vínculos asociados con la clase automáticamente. Las incoherencias de atributos detectadas se corrigen en todos los nombres de ruta de la lista generada.

Cómo se procesan las clases durante la eliminación de los paquetes

Los objetos se suprimen clase a clase. Las clases que existen para un paquete pero que no aparecen en el parámetro `CLASSES` se suprimen en primer lugar (por ejemplo, un objeto instalado con el comando `installf)`. Las clases que aparecen en el parámetro `CLASSES` se suprimen en el orden inverso. La clase `none` siempre se suprime en último lugar. A continuación se describen las acciones del sistema que se producen cuando se suprime una clase:

1. El comando `pkgrm` crea una lista de nombres de ruta.

   El comando `pkgrm` crea una lista de nombres de rutas instaladas que pertenecen a la clase indicada. Los nombres de rutas a los que hace referencia otro paquete se excluyen de la lista, a menos que el tipo de archivo sea `e`. Un tipo de archivo `e` significa que el archivo se debe editar tras la instalación o la supresión.

   Si el paquete que se suprime había modificado archivos del tipo `e` durante la instalación, sólo debe suprimir las líneas que agregó. No suprima un archivo modificable que no esté vacío. Suprima las líneas que agregó el paquete.

2. Si no existe una secuencia de comandos de acción de clase, las nombres de rutas se suprimen.

   Si el paquete no tiene una secuencia de comandos de acción de clase de eliminación para la clase, se suprimen todos los nombres de rutas de la lista generados por el comando `pkgrm`.

   Nota – Los archivos del tipo `e` (modificable) no se asignan a una clase ni a una secuencia de comandos de acción de clase asociada. Estos archivos se suprimen en este momento, incluso si el nombre de ruta se comparte con otros paquetes.

   ---

3. Si existe una secuencia de comandos de acción de clase, se ejecutará.

   El comando `pkgrm` invoca la secuencia de comandos de acción de clase con entrada estándar para la secuencia de comandos que contiene la lista generada en el elemento 1.

4. El comando `pkgrm` lleva a cabo una auditoría.

   Después de ejecutar correctamente la secuencia de comandos de acción de clase, el comando `pkgrm` suprime las referencias a los nombres de ruta de la base de datos de paquetes a menos que otro paquete haga referencia a un nombre de ruta.

Secuencia de comandos de acción de clase

La secuencia de comandos de acción de clase define un conjunto de acciones que se ejecutarán durante la instalación o eliminación de un paquete. Las acciones se llevarán cabo en un grupo de nombres de ruta basados en su definición de clase. Consulte el Capítulo 5, “Casos prácticos de creación de paquetes”, si desea ver ejemplos de secuencias de comandos de acción de clase.

El nombre de una secuencia de comandos de acción de clase se basa en la clase en la que debe funcionar y si dichas operaciones deben producirse durante la instalación o la eliminación de paquetes. Los dos formatos de nombres aparecen en la tabla siguiente:

Formato de nombre	Descripción
`i.`*class*	Realiza operacinoes en nombres de ruta en la clase indicada durante la instalación de paquetes
`r.`*class*	Realiza operaciones en nombres de ruta en la clase indicada durante la eliminación de paquetes

Por ejemplo, el nombre de la secuencia de comandos de instalación de una clase llamada `manpage` sería `i.manpage`. La secuencia de comandos de eliminación se llamaría `r.manpage`.

Nota – Este formato de nombre de archivos no se utiliza con los archivos que pertenecen a las clases de sistema `sed`, `awk` o `build`. Para obtener más información sobre estas clases especiales, consulte "Clases de sistema especiales" en la página 75.

Comportamientos de las secuencias de comandos de acción de clase

- Las secuencias de comandos de acción de clase se ejecutan como `uid=root` y `gid=other`.
- Se ejecuta una secuencia de comandos para todos los archivos de la clase determinada en el volumen actual.
- Los comandos `pkgadd` y `pkgrm` crean una lista de todos los objetos que aparecen en el archivo `pkgmap` que pertenezcan a la clase. Como resultado, una secuencia de comandos de acción de clase sólo puede actuar cuando se hayan definido nombres de rutas en el archivo `pkgmap` que pertenezcan a una clase determinada.
- Si una secuencia de comandos de acción de clase se ejecuta por última vez (es decir, no hay más archivos que pertenezcan a esa clase), lo hace con el argumento de palabra clave `ENDOFCLASS`.
- No se permite la interacción con el administrador durante la ejecución de una secuencia de comandos de procedimiento.

Normativa de diseño para secuencias de comandos de acción de clase

- Si un paquete abarca más de un volumen, la secuencia de comandos de acción de clase se ejecuta una vez para cada volumen que contenga al menos un archivo que pertenezca a una clase. En consecuencia, cada secuencia de comandos debe poder ejecutarse más de una vez. Esto significa que la ejecución de una secuencia de comandos un determinado número de veces con la misma entrada debe producir los mismos resultados que la ejecución de la secuencia de comandos sólo una vez.

- Si un archivo forma parte de una clase que cuenta con una secuencia de comandos de acción de clase, dicha secuencia debe instalar el archivo. El comando `pkgadd` no instala los archivos por los que existe una secuencia de comandos de acción de clase, aunque verifica la instalación.
- Una secuencia de comandos de acción de clase nunca debe agregar, suprimir ni modificar un nombre de ruta o atributo de sistema que no aparezca en la lista generada por el comando `pkgadd`. Para obtener más información sobre esta lista, consulte el elemento 1 en "Cómo se procesan las clases durante la instalación de los paquetes" en la página 72.
- Cuando la secuencia de comandos vea el argumento `ENDOFCLASS`, coloque las acciones de posproceso como la limpieza en la secuencia de comandos.
- Toda la interacción con el administrador se restringe a la secuencia de comandos `request`. No intente obtener información del administrador mediante una secuencia de comandos de acción de clase.

Clases de sistema especiales

El sistema ofrece cuatro clases especiales:

- La clase `sed`

  Proporciona un método para usar las instrucciones `sed` con el fin de editar archivos tras la instalación y eliminación de paquetes.
- La clase `awk`

  Proporciona un método para usar las instrucciones `awk` con el fin de editar archivos tras la instalación y eliminación de paquetes.
- La clase `build`

  Proporciona un método para crear o modificar dinámicamente un archivo usando comandos del shell Bourne.
- La clase `preserve`

  Ofrece un método para conservar archivos que no deben sobrescribirse en futuras instalaciones de paquetes.
- La clase `manifest`

  Ofrece desinstalación e instalación automatizadas de los servicios SMF (Utilidad de gestión de servicios) asociados a un manifiesto. La clase `manifest` se debe utilizar para todos los manifiestos SMF de un paquete.

Si varios archivos de un paquete requieren un procesamiento especial que se puede definir completamente a través de los comandos `sed`, `awk` o `sh`, la instalación es más rápida si se usan clases de sistemas en lugar de varias clases y sus secuencias de comandos de acción de clase correspondientes.

Secuencia de comandos de clase sed

La clase `sed` ofrece un método para modificar un objeto del sistema de destino. La secuencia de comandos de acción de clase `sed` se ejecuta automáticamente en la instalación si existe un archivo que pertenece a la clase `sed`. El nombre de la secuencia de comandos de acción de clase `sed` debe ser el mismo que el del archivo donde se ejecutan las instrucciones.

Una secuencia de comandos de acción de clase `sed` entrega instrucciones `sed` en el formato siguiente:

Dos comandos indican cuándo se deben ejecutar las instrucciones. Se ejecutan las instrucciones `sed` que siguen al comando `!install` durante la instalación del paquete. Se ejecutan las instrucciones `sed` que siguen al comando `!remove` durante la eliminación de paquetes. No importa el orden en el que estos comandos se usan en el archivo.

Para obtener más información sobre las instrucciones `sed` consulte la página de comando man `sed(1)` Si desea ver ejemplos de las secuencias de comandos de acción de clase `sed`, consulte el Capítulo 5, “Casos prácticos de creación de paquetes”.

Secuencia de comandos de clase awk

La clase `awk` ofrece un método para modificar un objeto del sistema de destino. Las modificaciones se entregan como instrucciones `awk` en una secuencia de comandos de acción de clase `awk`.

La secuencia de comandos de acción de clase `awk` se ejecuta automáticamente en la instalación si existe un archivo que pertenece a la clase `awk`. Dicho archivo contiene instrucciones para la secuencia de comandos de clase `awk` con el formato siguiente:

Dos comandos indican cuándo se deben ejecutar las instrucciones. Se ejecutan las instrucciones `awk` que siguen al comando `!install` durante la instalación del paquete. Durante la eliminación del paquete se ejecutan las instrucciones que siguen al comando `!remove`. Estos comandos se pueden usar en cualquier orden.

El nombre de la secuencia de comandos de acción de clase `awk` debe ser el mismo que el del archivo donde se ejecutan las instrucciones.

El archivo que se debe modificar se utiliza como entrada al comando `awk` y la salida de la secuencia de comandos sustituye por último al objeto original. Es posible que las variables de entorno no pasen al comando `awk` con esta sintaxis.

Para obtener más información sobre las instrucciones `awk`, consulte la página de comando man `awk(1)`.

Secuencia de comandos de clase build

La clase `build` crea o modifica un archivo de objeto de paquete mediante la ejecución de las instrucciones del shell Bourne. Estas instrucciones se entregan como objeto del paquete. Las instrucciones se ejecutan automáticamente en la instalación si el objeto del paquete pertenece a la clase `build`.

El nombre de la secuencia de comandos de acción de clase `build` debe ser el mismo que el del archivo donde se ejecutan las instrucciones. El nombre también debe ser ejecutable por parte del comando `sh`. La salida de la secuencia de comandos se convierte en la nueva versión del archivo al construirse o modificarse. Si la secuencia de comandos no produce salida alguna, el archivo no se crea ni se modifica. Por ello, la secuencia de comandos puede modificar o crear el archivo por sí mismo.

Por ejemplo, si un paquete entrega un archivo predeterminado, `/etc/randomtable`, y si éste no existe ya en el sistema de destino, la entrada del archivo `prototype` puede ser como la siguiente:

```
e build /etc/randomtable ? ? ?
```

El objeto del paquete, `/etc/randomtable`, puede tener el aspecto siguiente:

```
!install
# randomtable builder
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
        echo "/etc/randomtable is already in place.";
        else
        echo "# /etc/randomtable" > $PKG_INSTALL_ROOT/etc/randomtable
        echo "1121554    # first random number" >> $PKG_INSTALL_ROOT/etc/randomtable
fi

!remove
# randomtable deconstructor
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
        # the file can be removed if it's unchanged
        if [ egrep "first random number" $PKG_INSTALL_ROOT/etc/randomtable ]; then
            rm $PKG_INSTALL_ROOT/etc/randomtable;
        fi
fi
```

Consulte el Capítulo 5, "Casos prácticos de creación de paquetes", si desea ver otro ejemplo mediante la clase `build`.

Secuencia de comandos de clase preserve

La clase `preserve` conserva un archivo de objeto de paquete determinando si un archivo existente debe sobrescribirse cuando se instala el paquete. Al usar una secuencia de comandos de clase `preserve` puede haber dos situaciones:

- Si el archivo que se va a instalar no existe ya en el directorio de destino, se instalará normalmente.

- Si el archivo que se va a instalar existe en el directorio de destino, aparece un mensaje para informar de que el archivo existe, por lo que no se instala.

El resultado de ambas situaciones se considera correcto por parte de la secuencia de comandos `preserve`. Sólo se produce un error en la segunda situación si el archivo no puede copiarse en el directorio de destino.

Desde Solaris 7 puede encontrarse la secuencia de comandos `i.preserve` y una copia suya, `i.CONFIG.prsv`, en el directorio `/usr/sadm/install/scripts` con las demás secuencias de comandos de acción de clase.

Modifique la secuencia de comandos para que incluya los nombres de archivos que desee conservar.

La secuencia de comandos de clase manifest

La clase `manifest` instala y desinstala automáticamente los servicios SMF (Utilidad de gestión de servicios) asociados a un manifiesto SMF. Si no está familiarizado con la utilidad de gestión de servicios, consulte el Capítulo 18, "Gestión de servicios (descripción general)" de *Administración de Oracle Solaris: administración básica* para obtener información sobre cómo usar la utilidad de gestión de servicios con el fin de gestionar servicios.

Todos los manifiestos de servicios de los paquetes se deben identificar con la clase `manifest`. Las secuencias de comandos de acción de clase que instalan y suprimen manifiestos de servicios se incluyen en el subsistema de empaquetado. Cuando se invoca `pkgadd`(1M), se importa el manifiesto de servicios. Cuando se invoca `pkgrm`(1M), se suprimen las instancias del manifiesto de servicios que están desactivadas. Los servicios del manifiesto que no tienen instancias restantes también se suprimen. Si se suministra la opción `-R` a `pkgadd`(1M) o `pkgrm`(1M), estas acciones de manifiesto de servicios se efectuarán cuando el sistema se reinicie la próxima vez con esa ruta raíz alternativa.

La porción siguiente de código de un archivo de información del paquete muestra el uso de la clase `manifest`.

```
# packaging files
i pkginfo
i copyright
i depend
i preinstall
i postinstall
#
# source locations relative to the prototype file
#
d none var 0755 root sys
d none var/svc 0755 root sys
d none var/svc/manifest 0755 root sys
d none var/svc/manifest/network 0755 root sys
d none var/svc/manifest/network/rpc 0755 root sys
f manifest var/svc/manifest/network/rpc/smserver.xml 0444 root sys
```

Nota – No incluya los archivos `i.manifest` y `r.manifest` en los paquetes, dado que estas secuencias de comandos de acción de clase forman parte del sistema operativo Oracle Solaris y varían de una versión a otra. La inclusión de estos archivos hace que los paquetes sean menos portátiles entre diferentes versiones de Oracle Solaris.

▼ Cómo escribir secuencias de comandos de acción de clase

1. **Convierta el directorio que contiene sus archivos de información en el directorio de trabajo actual.**

2. **Asigne a los objetos del paquete del archivo `prototype` los nombres de clase deseados.**

   Por ejemplo, la asignación de objetos a las clases `application` y `manpage` tendría el aspecto siguiente:

   ```
   f manpage /usr/share/man/manl/myappl.1l
   f application /usr/bin/myappl
   ```

3. **Modifique el parámetro `CLASSES` del archivo `pkginfo` para que contenga los nombres de clase que desee usar en el paquete.**

   Por ejemplo, las entradas para las clases `application` y `manpage` tendrían el aspecto siguiente:

   ```
   CLASSES=manpage application none
   ```

   Nota – La clase `none` siempre se instala en primer lugar y se suprime en último lugar, independientemente de dónde aparezca en la definición del parámetro `CLASSES`.

4. **Si crea una secuencia de comandos de acción de clase para un archivo que pertenezca a la clase `sed`, `awk` o `build`, convierta el directorio que contiene el objeto del paquete en el directorio de trabajo actual.**

5. **Cree las secuencias de comandos de acción de clase o los objetos del paquete (para los archivos que pertenecen a la clase `sed`, `awk`, o `build`).**

   Por ejemplo, una secuencia de comandos de instalación de una clase llamada `application` recibiría el nombre de `i.application`, y una secuencia de comandos de eliminación recibiría el nombre de `r.application`.

   Recuerde, si un archivo forma parte de una clase que cuenta con una secuencia de comandos de acción de clase, dicha secuencia debe instalar el archivo. El comando `pkgadd` no instala los archivos por los que existe una secuencia de comandos de acción de clase, aunque verifica la instalación. Si define una clase pero no entrega una secuencia de comandos de acción de clase, la

única acción llevada a cabo para esa clase es copiar componentes del medio de instalación al sistema de destino (el comportamiento predeterminado de `pkgadd`).

6 **Complete *una* de las tareas siguientes:**

- Si *no* ha creado el archivo `prototype`, complete el procedimiento "Cómo crear un archivo `prototype` mediante el comando `pkgproto`" en la página 43 y vaya al Paso 7.
- Si ya ha creado su archivo `prototype`, edítelo y agregue una entrada para cada secuencia de comandos de instalación que acabe de crear.

7 **Construya su paquete.**

Consulte "Cómo construir un paquete" en la página 46 si fuera necesario.

Más información Paso siguiente

Después de construir el paquete, instálelo para confirmar que se instala correctamente y compruebe su integridad. En el Capítulo 4, "Verificación y transferencia de un paquete" se explica cómo hacerlo y se ofrecen instrucciones paso a paso para transferir su paquete verificado a un medio de distribución.

Creación de paquetes firmados

El proceso de creación de paquetes firmados implica varios pasos y requiere cierta comprensión de los nuevos conceptos y la terminología. En esta sección se ofrece información sobre paquetes firmados, su terminología, así como información sobre administración de certificados. Esta sección también ofrece procedimientos detallados sobre cómo crear un paquete firmado.

Paquetes firmados

Un paquete firmado tiene formato de flujo normal que cuenta con una firma digital (firma digital PKCS7 con código PEM que se define a continuación) que verifica lo siguiente:

- El paquete procede de la entidad que lo firmó
- La entidad la firmó
- El paquete no se ha modificado desde que la entidad lo firmó
- La entidad que lo firmó es una entidad de confianza

La firma es la única diferencia entre un paquete firmado y otro sin firmar. Un paquete firmado es compatible en el nivel binario con un paquete sin firmar. Por ello se puede usar un paquete firmado con versiones anteriores de las herramientas de creación de paquetes. Sin embargo, la firma se ignora en este caso.

La tecnología de creación de paquetes firmados presenta nueva terminología y abreviaturas que se describen en la tabla siguiente.

Término	Definición
ASN.1	Notación de sintaxis abstracta 1: modo de expresar objetos abstractos. Por ejemplo, ASN.1 define un certificado de clave pública, todos los objetos que componen el certificado, y el orden en el que los objetos se recogen. Sin embargo, ASN.1 no especifica cómo los objetos se serializan para el almacenamiento o la transmisión.
X.509	Recomendación ITU-T X.509: especifica la sintaxis del certificado de clave pública X.509 ampliamente adoptado.
DER	Normas de codificación distinguidas: representación binaria de un objeto ASN.1 y define cómo un objeto ASN.1 se serializa para el almacenamiento o la transmisión en los entornos computacionales.
PEM	Mensaje de privacidad mejorada: forma de codificar un archivo (en DER u otro formato binario) mediante codificación base 64 y algunos encabezados optativos. PEM se utilizaba inicialmente para codificar mensajes de correo electrónico de tipo MIME. PEM también se usa ampliamente para codificar certificados y claves privadas en un archivo que existe en un sistema de archivos o en un mensaje de correo electrónico.
PKCS7	Estándar de criptografía de clave pública n.º 7: este estándar describe una sintaxis general para datos a los que se ha aplicado criptografía, como firmas y sobres digitales. Un paquete firmado contiene una firma PKCS7 incorporada. Esta firma contiene como mínimo la recopilación cifrada del paquete, junto con el certificado de clave pública X.509 del firmante. El paquete firmado también puede contener certificados en cadena. Se pueden usar los certificados en cadena al formar una cadena de confianza del certificado del firmante a un certificado de confianza almacenado localmente.
PKCS12	Estándar de criptografía de clave pública n.º 12: este estándar describe una sintaxis para almacenar objetos criptográficos en el disco. El almacén de claves de paquetes se mantiene en este formato.
Almacén de claves de paquetes	Repositorio de certificados y claves que las herramientas de paquetes pueden consultar.

Gestión de certificados

Antes de crear un paquete firmado, debe tener un almacén de claves de paquetes. Este almacén de claves de paquetes contiene certificados en forma de objetos. Hay dos tipos de objetos en un almacén de claves de paquetes:

Certificado de confianza
: Un certificado de confianza contiene un único certificado de clave pública que pertenece a otra entidad. El certificado de confianza se nombra de este modo porque el propietario del almacén de claves

	confía en que la clave pública del certificado pertenece en realidad a la identidad indicada por el "sujeto" (propietario) del certificado. El emisor del certificado da fe de esta confianza al firmar el certificado. Los certificados de confianza se usan al verificar firmas y al iniciar una conexión con un servidor (SSL) seguro.
Clave de usuario	Una clave de usuario contiene información de clave criptográfica sensible. Esta información se almacena con un formato protegido para impedir el acceso no autorizado. Una clave de usuario se compone de la clave privada del usuario y el certificado de clave pública que se corresponde con la clave privada. Las claves de usuario se usan al crear un paquete firmado.

De forma predeterminada, el almacén de claves de paquetes se encuentra en el directorio `/var/sadm/security`. Cada uno de los usuarios también puede tener su propio almacén de claves guardado de forma predeterminada en el directorio `$HOME/.pkg/security`.

En el disco, un almacén de claves de paquetes puede estar en dos formatos: formato de varios archivos y formato de un único archivo. Un formato de varios archivos almacena sus objetos en varios archivos. Cada tipo de objeto se almacena en un archivo aparte. Todos estos archivos se deben cifrar mediante las mismas frases de contraseña. Un almacén de claves de un único archivo guarda todos sus objetos en un único archivo del sistema de archivos.

La utilidad principal que se usa para administrar los certificados y el almacén de claves de paquetes es el comando `pkgadm`. Las subsecciones siguientes describen las tareas más habituales utilizadas par administrar el almacén de claves de paquetes.

Agregación de certificados de confianza al almacén de claves de paquetes

Se puede agregar un certificado de confianza al almacén de claves de paquetes mediante el comando `pkgadm`. El certificado puede tener formato PEM o DER. Por ejemplo:

```
$ pkgadm addcert -t /tmp/mytrustedcert.pem
```

En este ejemplo, el certificado con formato PEM llamado `mytrustedcert.pem` se agrega al almacén de claves de paquetes.

Agregación de un certificado de usuario y de una clave privada al almacén de claves de paquetes

El comando `pkgadm` no genera certificados de usuarios o claves privadas. Los certificados de usuario y las claves privadas se obtienen normalmente de una autoridad de certificados, como

Verisign. O bien, se generan localmente como certificado autofirmado. Tras obtener la clave y el certificado, se pueden importar al almacén de claves de paquetes mediante el comando `pkgadm`. Por ejemplo:

```
pkgadm addcert -n myname -e /tmp/myprivkey.pem /tmp/mypubcert.pem
```

En este ejemplo se usan las opciones siguientes:

`-n` *myname*	Identifica a la entidad (*myname*) en el almacén de claves de paquetes donde desea trabajar. La entidad *myname* se convierte en alias en el que se almacenan los objetos.
`-e /tmp/`*myprivkey.pem*	Especifica el archivo que contiene la clave privada. En este caso, el archivo es *myprivkey.pem* y se encuentra en el directorio `/tmp`.
`/tmp/`*mypubcert.pem*	Especifica el archivo de certificado con formato PEM llamado *mypubcert.pem*.

Comprobación del contenido en el almacén de claves de paquetes

El comando `pkgadm` también se usa para ver el contenido del almacén de claves de paquetes. Por ejemplo:

```
$ pkgadm listcert
```

Este comando muestra los certificados de confianza y las claves privadas en el almacén de claves de paquetes.

Eliminación de certificados de confianza y claves privadas del almacén de claves de paquetes

El comando `pkgadm` se puede usar para suprimir certificados de confianza y claves privadas del almacén de claves de paquetes.

Cuando suprime certificados de usuarios, debe especificarse el alias del par certificado/clave. Por ejemplo:

```
$ pkgadm removecert -n myname
```

El alias del certificado es el nombre habitual del certificado que se puede identificar mediante el comando `pkgadm listcert`. Por ejemplo, este comando suprime un certificado de confianza de nombre Trusted CA Cert 1:

```
$ pkgadm removecert -n "Trusted CA Cert 1"
```

Nota – Si tiene un certificado de confianza y un certificado de usuario almacenados con el mismo alias se suprimen cuando especifica la opción `-n`.

Creación de paquetes firmados

El proceso de crear paquetes firmados implica tres pasos básicos:

1. Creación de un paquete con formato de directorio sin firmar.
2. Importación del certificado de firma, certificados de AC y clave privada en el almacén de claves de paquetes.
3. Firma del paquete del paso 1 con los certificados del paso 2.

Nota – Las herramientas de creación de paquetes no crean certificados. Estos certificados se deben obtener de una autoridad de certificados, como Verisign o Thawte.

Cada uno de los pasos para crear paquetes firmados se describe en los procedimientos siguientes.

▼ Cómo crear un paquete con formato de directorio sin firmar

El procedimiento para crear un paquete con formato de directorio sin firmar es el mismo que para crear un paquete normal, tal como se ha descrito previamente en este manual. El procedimiento siguiente describe el proceso de crear este paquete con formato de directorio sin firmar. Si necesita más información, consulte las secciones anteriores sobre la construcción de paquetes.

1 Cree el archivo `pkginfo`.

El archivo `pkginfo` debe tener el siguiente contenido básico:

```
PKG=SUNWfoo
BASEDIR=/
NAME=My Test Package
ARCH=sparc
VERSION=1.0.0
CATEGORY=application
```

2 Cree el archivo `prototype`.

El archivo `prototype` debe tener el siguiente contenido básico:

```
$cat prototype
i pkginfo
d none usr 0755 root sys
```

```
d none usr/bin 0755 root bin
f none usr/bin/myapp=/tmp/myroot/usr/bin/myapp 0644 root bin
```

3 Enumere el contenido del directorio de origen de objetos.

Por ejemplo:

```
$ ls -lR /tmp/myroot
```

La salida tendría un aspecto similar al siguiente:

```
/tmp/myroot:
total 16
drwxr-xr-x   3 abc      other        177 Jun  2 16:19 usr

/tmp/myroot/usr:
total 16
drwxr-xr-x   2 abc      other        179 Jun  2 16:19 bin

/tmp/myroot/usr/bin:
total 16
-rw-------   1 abc      other       1024 Jun  2 16:19 myapp
```

4 Cree el paquete sin firma.

```
pkgmk -d ‘pwd‘
```

La salida tendría un aspecto similar al siguiente:

```
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter <PSTAMP> set to "syrinx20030605115507"
WARNING: parameter <CLASSES> set to "none"
## Attempting to volumize 3 entries in pkgmap.
part  1 -- 84 blocks, 7 entries
## Packaging one part.
/tmp/SUNWfoo/pkgmap
/tmp/SUNWfoo/pkginfo
/tmp/SUNWfoo/reloc/usr/bin/myapp
## Validating control scripts.
## Packaging complete.
```

El paquete existe en el directorio actual.

▼ Cómo importar certificados al almacén de claves de paquetes

El certificado y la clave privada que se vayan a importar deben existir como certificado X.509 con codificación PEM o DER y clave privada. Asimismo, los certificados intermedios o de cadena que vinculen el certificado con firma al certificado de Autoridad de certificados se debe importar al almacén de claves de paquetes antes de que un paquete se pueda firmar.

Nota – Cada autoridad de certificados puede emitir certificados en diversos formatos. Para extraer los certificados y la clave privada del archivo PKCS12 a un archivo X.509 con codificación PEM (adecuado para importar al almacén de claves de paquetes), use una utilidad de conversión de software gratuito como OpenSSL.

Si la clave privada se cifra (lo cual sería normal), se le pedirá la frase de contraseña. Asimismo, se le pedirá una contraseña para proteger el almacén de claves de paquetes resultante. Si lo desea puede evitar el uso de una contraseña, pero con ello obtendrá un almacén de claves de paquetes sin cifrar.

El procedimiento siguiente describe cómo importar los certificados mediante el comando `pkgadm` una vez que el certificado tenga el formato adecuado.

1 Importe todos los certificados de Autoridad de certificados que se encuentren en el archivo de certificado X.509 con codificación PEM o DER.

Por ejemplo, para importar todos los certificados de Autoridad de certificados que se encuentren en el archivo `ca.pem`, debería escribir lo siguiente:

```
$ pkgadm addcert -k ~/mykeystore -ty ca.pem
```

La salida tendría un aspecto similar al siguiente:

```
Trusting certificate <VeriSign Class 1 CA Individual \
Subscriber-Persona Not Validated>
Trusting certificate </C=US/O=VeriSign, Inc./OU=Class 1 Public \
Primary Certification Authority
Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
For Verification: Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
Certificate(s) from <ca.pem> are now trusted
```

Con el fin de importar la clave de firma al almacén de claves de paquetes, debe proporcionar un alias que se utilice posteriormente al firmar el paquete. Este alias también se puede usar si desea suprimir la clave del almacén de claves de paquetes.

Por ejemplo, para importar la clave de firma del archivo `sign.pem`, debería escribir lo siguiente:

```
$ pkgadm addcert -k ~/mykeystore -n mycert sign.pem
```

La salida tendría un aspecto similar al siguiente:

```
Enter PEM passphrase:
Enter Keystore Password:
Successfully added Certificate <sign.pem> with alias <mycert>
```

2. **Compruebe que los certificados se encuentren en el almacén de claves de paquetes.**

   Por ejemplo, para ver los certificados del almacén de claves creados en el paso anterior, debería escribir lo siguiente:

   ```
   $ pkgadm listcert -k ~/mykeystore
   ```

▼ Cómo firmar el paquete

Cuando los certificados se importan al almacén de claves de paquetes puede firmar el paquete. La firma real del paquete se lleva a cabo mediante el comando `pkgtrans`.

- **Firme el paquete mediante el comando `pkgtrans`. Indique la ubicación del paquete sin firmar y el alias de la clave para firmar el paquete.**

  Por ejemplo, de acuerdo con los procedimientos anteriores, debería escribir lo siguiente para crear un paquete firmado llamado `SUNWfoo.signed`:

  ```
  $ pkgtrans -g -k ~/mykeystore -n mycert . ./SUNWfoo.signed SUNWfoo
  ```

  La salida de este comando tendría un aspecto similar al siguiente:

  ```
  Retrieving signing certificates from keystore </home/user/mykeystore>
  Enter keystore password:
  Generating digital signature for signer <Test User>
  Transferring <SUNWfoot> package instance
  ```

  El paquete firmado se crea en el archivo `SUNWfoo.signed` y se encuentra en el formato de flujo de paquete. Este paquete firmado es adecuado para copiarlo en un sitio web e instalarse mediante el comando `pkgadd` y un URL.

◆ ◆ ◆ CAPÍTULO 4

Verificación y transferencia de un paquete

En este capítulo se describe cómo verificar la integridad del paquete y transferirlo a un medio de distribución, como un disquete o un CD-ROM.

A continuación, se muestra una lista de la información general de este capítulo:

Verificación y transferencia de un paquete (mapa de tareas)

En la tabla siguiente, se describen los pasos que debe seguir con el fin de verificar la integridad del paquete y transferirlo a un medio de distribución.

TABLA 4–1 Verificación y transferencia de un paquete (mapa de tareas)

Tarea	Descripción	Instrucciones
1. Construir el paquete	Construya el paquete en el disco.	Capítulo 2, "Creación de un paquete"
2. Instalar el paquete	Pruebe la instalación del paquete y asegúrese de que se instala sin errores.	"Cómo instalar un paquete en un servidor o un sistema autónomo" en la página 91
3. Verificar la integridad del paquete	Use el comando `pkgchk` para verificar la integridad del paquete.	"Cómo verificar la integridad de un paquete" en la página 93
4. Obtener información adicional del paquete	*Opcional*. Use los comandos `pkginfo` y `pkgparam` para llevar a cabo la verificación específica del paquete.	"Visualización de información adicional sobre paquetes instalados" en la página 94

TABLA 4–1 Verificación y transferencia de un paquete (mapa de tareas) *(Continuación)*

Tarea	Descripción	Instrucciones
5. Suprimir el paquete instalado	Use el comando `pkgrm` para suprimir el paquete instalado del sistema.	"Cómo suprimir un paquete" en la página 99
6. Transferir el paquete a un medio de distribución	Use el comando `pkgtrans` para transferir el paquete (con formato de paquete) a un medio de distribución.	"Cómo transferir un paquete a un medio de distribución" en la página 100

Instalación de paquetes de software

Los paquetes de software se instalan mediante el comando `pkgadd`. Este comando transfiere el contenido de un paquete de software del medio de distribución o directorio y lo instala en un sistema.

En esta sección se ofrecen instrucciones básicas para instalar el paquete con el fin de verificar que se instala correctamente.

La base de datos de software de instalación

La información para todos los paquetes instalados en un sistema se mantiene en la base de datos de software de instalación. Hay una entrada para todos los objetos de un paquete, con información como el nombre del componente, dónde reside y su tipo. Una entrada contiene un registro del paquete al que pertenece un componente; otros paquetes que pueden hacer referencia al componente, e información como el nombre de la ruta, dónde reside el componente y de qué tipo es. Los comandos `pkgadd` y `pkgrm` agregan y eliminan automáticamente las entradas. Puede ver la información en la base de datos mediante los comandos `pkgchk` y `pkginfo`.

Se asocian dos tipos de información a cada componente de paquete. La información del atributo describe el componente en sí mismo. Por ejemplo, los permisos de acceso del componente, ID de propietario e ID de grupo son información de atributos. La información de contenido describe el contenido del componente, como el tamaño del archivo y la hora de la última modificación.

La base de datos de software de instalación hace un seguimiento del estado del paquete. Una paquete se puede instalar completamente (ha completado el proceso de instalación), o bien parcialmente (no ha completado el proceso de instalación).

Si un paquete se instala parcialmente, es posible que partes de un paquete se hayan instalado antes de que se terminara la instalación; por ello, una parte del paquete está instalada y registrada en la base de datos, y una parte no. Cuando vuelve a instalar el paquete, se le indica que comience en el punto en que se detuvo la instalación porque el comando `pkgadd` puede acceder a la base de datos y detectar qué partes se han instalado ya. También puede suprimir las partes que se han instalado, de acuerdo con la información de la base de datos de software de instalación mediante el comando `pkgrm`.

Interactuación con el comando pkgadd

Si el comando pkgadd encuentra un problema, en primer lugar comprueba las instrucciones en el archivo de administración de la instalación. Consulte admin(4) para obtener más información. Si no se encuentran instrucciones, o bien si el parámetro pertinente del archivo de administración está configurado en ask, pkgadd muestra un mensaje que describe el problema y solicita una respuesta. El indicador es generalmente Do you want to continue with this installation? (¿Desea continuar con la instalación?). Debe responder con yes (sí), no o quit (salir).

Si ha especificado más de un paquete, no detiene la instalación del paquete pero pkgadd continúa con la instalación de otros paquetes. quit indica que pkgadd debe detener la instalación de todos los paquetes.

Instalación de paquetes en sistemas autónomos o servidores en un entorno homogéneo

En esta sección, se describe cómo instalar paquetes en un sistema autónomo o un servidor en un entorno homogéneo.

▼ Cómo instalar un paquete en un servidor o un sistema autónomo

1. **Cree el paquete.**

   Consulte "Creación de un paquete" en la página 45, si fuera necesario.

2. **Inicie la sesión como superusuario.**

3. **Agregue el paquete de software al sistema.**

   ```
   # pkgadd -d device-name [pkg-abbrev...]
   ```

-d *device-name*	Especifica la ubicación del paquete. Tenga en cuenta que *device-name* puede ser un nombre de ruta de directorio completo o los identificadores de una cinta, disquete o disco extraíble.
pkg-abbrev	Es el nombre de uno o más paquetes (separados por espacios) que se agregarán. Si se omite, pkgadd instala todos los paquetes disponibles.

Ejemplo 4–1 Instalación de paquetes en servidores y sistemas autónomos

Para instalar un paquete de software de nombre `pkgA` a partir de un dispositivo de cinta de nombre `/dev/rmt/0`, debería especificar el comando siguiente:

```
# pkgadd -d /dev/rmt/0 pkgA
```

También puede instalar varios paquetes al mismo tiempo, siempre que separe sus nombres con un espacio, de este modo:

```
# pkgadd -d /dev/rmt/0 pkgA pkgB pkgC
```

Si no nombra el dispositivo en el que reside el paquete, el comando comprueba el directorio de cola predeterminado (`/var/spool/pkg`). Si el paquete no está allí, la instalación falla.

Véase también Si está preparado para ir a la tarea siguiente, consulte "Cómo verificar la integridad de un paquete" en la página 93.

Verificación de la integridad de un paquete

El comando `pkgchk` permite comprobar la integridad de los paquetes, ya sea si están instalados en un sistema o tienen formato de paquete (listo para instalarse con el comando `pkgadd`). Confirma la estructura del paquete o los archivos instalados y los directorios, o bien muestra información sobre objetos del paquete. El comando `pkgchk` puede enumerar o comprobar lo siguiente:

- Las secuencias de comandos de instalación del paquete.
- El contenido o los atributos de los objetos instalados actualmente en el sistema.
- El contenido de un paquete sin instalar en cola.
- El contenido o los atributos, o ambos, de los objetos descritos en el archivo `pkgmap` especificado.

Para obtener más información sobre este comando, consulte `pkgchk(1M)`.

El comando `pkgchk` lleva a cabo dos tipos de comprobaciones. Comprueba los atributos del archivo (los permisos y la propiedad de un archivo, así como los números principales o secundarios del bloque o los dispositivos especiales de caracteres) y el contenido de los archivos (el tamaño, la suma de comprobación y la fecha de modificación). De forma predeterminada, el comando comprueba los atributos y el contenido de los archivos.

El comando `pkgchk` también compara los atributos y el contenido de los archivos del paquete instalado con la base de datos de software de instalación. Las entradas relacionadas con un paquete pueden haber cambiado desde el tiempo de la instalación; por ejemplo, puede que otro paquete haya cambiado un componente del paquete. La base de datos refleja dicho cambio.

▼ Cómo verificar la integridad de un paquete

1 Instale el paquete.

Consulte "Cómo instalar un paquete en un servidor o un sistema autónomo" en la página 91 si fuera necesario.

2 Verifique la integridad del paquete.

```
# pkgchk [-v] [-R root-path] [pkg-abbrev...]
```

`-v`	Muestra los archivos a medida que se procesan.
`-R` *root-path*	Especifica la ubicación del sistema de archivos raíz del sistema cliente.
pkg-abbrev	Es el nombre de uno o más paquetes (separados por espacios) que se comprobarán. Si se omite, `pkgchk` comprueba todos los paquetes disponibles.

Ejemplo 4–2 Verificación de la integridad de un paquete

Este ejemplo muestra el comando que debe usar para verificar la integridad de un paquete instalado.

```
$ pkgchk pkg-abbrev
$
```

Si hay errores, el comando `pkgchk` los imprime. De lo contrario, no se imprime nada y devuelve un código de salida de 0. Si no proporciona una abreviación de paquete, comprobará todos los paquetes del sistema.

Si lo desea, puede usar la opción `-v`, que imprimirá una lista de archivos del paquete si no hay errores. Por ejemplo:

```
$ pkgchk -v SUNWcadap
/opt/SUNWcadap
/opt/SUNWcadap/demo
/opt/SUNWcadap/demo/file1
/opt/SUNWcadap/lib
/opt/SUNWcadap/lib/file2
/opt/SUNWcadap/man
/opt/SUNWcadap/man/man1
/opt/SUNWcadap/man/man1/file3.1
/opt/SUNWcadap/man/man1/file4.1
/opt/SUNWcadap/man/windex
/opt/SUNWcadap/srcfiles
/opt/SUNWcadap/srcfiles/file5
/opt/SUNWcadap/srcfiles/file6
$
```

Si necesita verificar un paquete que esté instalado en el sistema de archivos raíz de un sistema cliente, use este comando:

```
$ pkgchk -v -R root-path pkg-abbrev
```

Véase también Si está preparado para ir a la tarea siguiente, consulte "Cómo obtener información con el comando pkginfo" en la página 98.

Visualización de información adicional sobre paquetes instalados

Puede usar los otros dos comandos para mostrar información sobre los paquetes instalados:

- El comando `pkgparam` muestra los valores de los parámetros.
- El comando `pkginfo` muestra información de la base de datos de software de instalación.

El comando `pkgparam`

El comando `pkgparam` permite mostrar los valores asociados a los parámetros que especificó en la línea de comandos. Los valores de un paquete específico se recuperan del archivo `pkginfo` o del archivo que indique. Se muestra un valor de parámetro por línea. Puede mostrar los valores solamente o los parámetros y sus valores.

▼ Cómo obtener información con el comando `pkgparam`

1 Instale el paquete.

Consulte "Cómo instalar un paquete en un servidor o un sistema autónomo" en la página 91 si fuera necesario.

2 Muestre información adicional sobre el paquete.

```
# pkgparam [-v] pkg-abbrev [param...]
```

`-v`	Muestra el nombre del parámetro y su valor.
pkg-abbrev	Es el nombre de un paquete específico.
param	Especifica uno o más parámetros cuyos valores se muestran.

Ejemplo 4–3 Obtención de información con el comando `pkgparam`

Por ejemplo, para mostrar valores solamente, use este comando.

```
$ pkgparam SUNWcadap
none
/opt
```

```
US/Mountain
/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin
/usr/sadm/sysadm
SUNWcadap
Chip designers need CAD application software to design abc
chips.  Runs only on xyz hardware and is installed in the usr
partition.
system
release 1.0
SPARC
venus990706083849
SUNWcadap
/var/sadm/pkg/SUNWcadap/save
Jul 7 1999 09:58
$
```

Para mostrar parámetros y sus valores, use el comando siguiente.

```
$ pkgparam -v SUNWcadap
pkgparam -v SUNWcadap
CLASSES='none'
BASEDIR='/opt'
TZ='US/Mountain'
PATH='/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin'
OAMBASE='/usr/sadm/sysadm'
PKG='SUNWcadap'
NAME='Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.'
CATEGORY='system'
VERSION='release 1.0'
ARCH='SPARC'
PSTAMP='venus990706083849'
PKGINST='SUNWcadap'
PKGSAV='/var/sadm/pkg/SUNWcadap/save'
INSTDATE='Jul 7 1999 09:58'
$
```

O bien, si desea mostrar el valor de un parámetro específico, use este formato:

```
$ pkgparam SUNWcadap BASEDIR
/opt
$
```

Para obtener más información, consulte `pkgparam`(1).

Véase también Si está listo para ir a la tarea siguiente, consulte "Cómo suprimir un paquete" en la página 99.

El comando `pkginfo`

Puede mostrar información sobre paquetes instalados con el comando `pkginfo`. Este comando tiene varias opciones que permiten personalizar tanto el formato como el contenido de lo que se visualiza.

Puede solicitar información sobre el número de instancias de paquetes que desee.

Visualización predeterminada de `pkginfo`

Si el comando `pkginfo` se ejecuta sin opciones, muestra la categoría, la instancia de paquete y el nombre de todos los paquetes que se han instalado completamente en el sistema. La pantalla se organiza por categorías, tal como se muestra en el ejemplo siguiente.

```
$ pkginfo
.
.
.
system      SUNWinst       Install Software
system      SUNWipc        Interprocess Communications
system      SUNWisolc      XSH4 conversion for ISO Latin character sets
application SUNWkcspf      KCMS Optional Profiles
application SUNWkcspg      KCMS Programmers Environment
application SUNWkcsrt      KCMS Runtime Environment
.
.
.
$
```

Personalización del formato de la pantalla de `pkginfo`

Puede ver la pantalla de `pkginfo` con tres formatos diferentes: corto, extraído y largo.

El formato corto es el predeterminado. Sólo muestra la categoría, la abreviatura del paquete y el nombre completo del paquete, tal como se indica en "Visualización predeterminada de `pkginfo`" en la página 96.

El formato extraído muestra la abreviatura, el nombre y la arquitectura del paquete (si están disponibles), así como su versión (si está disponible). Use la opción `-x` para solicitar el formato extraído tal como aparece en el ejemplo siguiente.

```
$ pkginfo -x
.
.
.
SUNWipc         Interprocess Communications
                (sparc) 11.8.0,REV=1999.08.20.12.37
SUNWisolc       XSH4 conversion for ISO Latin character sets
                (sparc) 1.0,REV=1999.07.10.10.10
SUNWkcspf       KCMS Optional Profiles
                (sparc) 1.1.2,REV=1.5
SUNWkcspg       KCMS Programmers Environment
                (sparc) 1.1.2,REV=1.5
.
.
.
$
```

El uso de la opción `-l` produce una pantalla de formato largo donde se muestra toda la información disponible de un paquete, como en el ejemplo anterior.

```
$ pkginfo -l SUNWcadap
   PKGINST:  SUNWcadap
      NAME:  Chip designers need CAD application software to
design abc chips.  Runs only on xyz hardware and is installed
in the usr partition.
  CATEGORY:  system
      ARCH:  SPARC
   VERSION:  release 1.0
   BASEDIR:  /opt
    PSTAMP:  system980706083849
  INSTDATE:  Jul 7 1999 09:58
    STATUS:  completely installed
     FILES:     13 installed pathnames
                 6 directories
                 3 executables
              3121 blocks used (approx)
$
```

Descripciones de parámetros para el formato largo de `pkginfo`

En la tabla siguiente, se describen los parámetros de paquetes que se pueden mostrar para cada paquete. Un parámetro y su valor aparecen solamente cuando el parámetro tiene un valor asignado.

TABLA 4–2 Parámetros de paquetes

Parámetro	Descripción
ARCH	La arquitectura admitida por este paquete.
BASEDIR	El directorio base donde reside el paquete de software (se muestra si el paquete es reubicable).
CATEGORY	Las categorías de software de las que este paquete es miembro (por ejemplo, `system` o `application`).
CLASSES	Una lista de clases definida para un paquete. El orden de la lista determina el orden en que se instalan las clases. Las clases enumeradas en primer lugar se instalan primero (medio a medio). Este parámetro se puede modificar mediante la secuencia de comandos de solicitud.
DESC	Texto que describe el paquete.
EMAIL	La dirección de correo electrónico para las consultas del usuario.
HOTLINE	Información sobre cómo recibir ayuda al instante sobre este paquete.
INTONLY	Indica que el paquete sólo debe instalarse interactivamente cuando se configura con un valor diferente de NULL.
ISTATES	Una lista de estados de ejecución posibles para la instalación de paquetes (por ejemplo, `S s 1`).
MAXINST	Número máximo de instancias de paquetes que se deben permitir en una máquina al mismo tiempo. De forma predeterminada, sólo se permite una instancia de un paquete.

TABLA 4–2 Parámetros de paquetes *(Continuación)*

Parámetro	Descripción
NAME	Nombre del paquete, generalmente texto que describe la abreviatura del paquete.
ORDER	Lista de clases que define el orden en que se deben colocar en el medio. Utilizado por el comando `pkgmk` en la creación del paquete. Las clases que no están definidas en este parámetro se colocan en el medio mediante procedimientos de ordenación estándar.
PKGINST	Abreviatura del paquete que se está instalando.
PSTAMP	Marca de producción de este paquete.
RSTATES	Lista de estados de ejecución posibles para la eliminación de paquetes (por ejemplo, `S s 1`).
ULIMIT	Si está configurado, este parámetro se pasa como argumento al comando `ulimit`, que establece el tamaño máximo de un archivo durante la instalación. Esto se aplica solamente a archivos creados por secuencias de comandos de procedimientos.
VENDOR	Nombre del distribuidor que proporcionó el paquete de software.
VERSION	Versión de este paquete.
VSTOCK	Número de existencias proporcionado por el distribuidor.

Para obtener información detallada sobre el comando `pkginfo`, consulte la página del comando man `pkginfo(1)`.

▼ Cómo obtener información con el comando `pkginfo`

1 Instale el paquete.

Consulte "Cómo instalar un paquete en un servidor o un sistema autónomo" en la página 91 si fuera necesario.

2 Muestre información adicional sobre el paquete.

```
# pkginfo [-x | -l] [pkg-abbrev]
```

`-x`	Muestra información del paquete en formato extraído.
`-l`	Muestra información del paquete en formato largo.
pkg-abbrev	Es el nombre de un paquete específico. Si se omite, el comando `pkginfo` muestra información sobre todos los paquetes instalados, en el formato predeterminado.

Más información Paso siguiente

Si está listo para ir a la tarea siguiente, consulte "Cómo suprimir un paquete" en la página 99.

Eliminación de un paquete

Debido a que el comando `pkgrm` actualiza la información en la base de datos de productos de software, es importante, si elimina un paquete, usar el comando `pkgrm`, aunque esté tentado a usar el comando `rm` en su lugar. Por ejemplo, podría usar el comando `rm` para eliminar un archivo ejecutable binario, aunque no es lo mismo que usar `pkgrm` para eliminar el paquete de software que incluye dicho ejecutable binario. El uso del comando `rm` para suprimir archivos de un paquete dañará la base de datos de productos de software. Si en realidad sólo desea eliminar un archivo, puede usar el comando `removef`, que actualizará la base de datos de productos de software correctamente.

▼ Cómo suprimir un paquete

1 **Inicie sesión en el sistema como superusuario.**

2 **Elimine un paquete instalado.**

```
# pkgrm pkg-abbrev ...
```

pkg-abbrev	Es el nombre de uno o más paquetes (separados por espacios). Si se omite, `pkgrm` suprime todos los paquetes disponibles.

3 **Verifique que el paquete se haya suprimido satisfactoriamente con el comando `pkginfo`.**

```
$ pkginfo | egrep pkg-abbrev
```

Si se ha instalado *pkg-abbrev*, el comando `pkginfo` devuelve una línea de información al respecto. De lo contrario, `pkginfo` devuelve el indicador del sistema.

Transferencia de un paquete a un medio de distribución

El comando `pkgtrans` mueve los paquetes y lleva a cabo traducciones de formato de los paquetes. Puede usar el comando `pkgtrans` para llevar a cabo las traducciones siguientes de un paquete instalable:

- Formato de sistema de archivos a formato de flujo de datos
- Formato de flujo de datos a formato de sistema de archivos
- Un formato de sistema de archivos a otro formato de sistema de archivos

▼ Cómo transferir un paquete a un medio de distribución

1 **Construya el paquete; cree un paquete con formato de directorio, si no lo ha hecho todavía.**

Para obtener más información, consulte "Cómo construir un paquete" en la página 46.

2 **Instale el paquete para verificar que se instale correctamente.**

Consulte "Cómo instalar un paquete en un servidor o un sistema autónomo" en la página 91 si fuera necesario.

3 **Verifique la integridad del paquete.**

Consulte "Cómo verificar la integridad de un paquete" en la página 93, "Cómo obtener información con el comando `pkginfo`" en la página 98 y "Cómo obtener información con el comando `pkgparam`" en la página 94, si fuera necesario.

4 **Quite el paquete instalado del sistema.**

Consulte "Cómo suprimir un paquete" en la página 99, si fuera necesario.

5 **Transfiera el paquete (con formato de paquete) a un medio de distribución.**

Para llevar a cabo una traducción básica, ejecute el comando siguiente:

```
$ pkgtrans device1 device2 [pkg-abbrev...]
```

device1	Es el nombre del dispositivo donde reside actualmente el paquete.
device2	Es el nombre del dispositivo en el que se escribirá el paquete traducido.
[*pkg-abbrev*]	Es una o más abreviaturas de paquetes.

Si no se proporcionan nombres de paquetes, todos los paquetes que se encuentran en el *device1* se traducen y se escriben en el *device2*.

Nota – Si más de una instancia de un paquete reside en el *device1*, debe usar un identificador de instancias para el paquete. Para obtener una descripción de un identificador de paquetes, consulte "Definición de la instancia de un paquete" en la página 27. Cuando una instancia del paquete que se está traduciendo ya existe en el *device2*, el comando `pkgtrans` no lleva a cabo la traducción. Puede usar la opción `-o` para indicar al comando `pkgtrans` que sobrescriba las instancias del dispositivo de destino y la opción `-n` para indicarle que cree una nueva instancia si ya existe una. Tenga en cuenta que esta comprobación no se aplica cuando el *device2* admite un formato de flujo de datos.

Más información

Paso siguiente

En este momento ha completado todos los pasos necesarios para diseñar, construir, verificar y transferir el paquete. Si está interesado en consultar algunos casos prácticos, consulte Capítulo 5, "Casos prácticos de creación de paquetes". Si está interesado en ideas de diseño de paquetes avanzados, consulte el Capítulo 6, "Técnicas avanzadas para la creación de paquetes".

◆ ◆ ◆ CAPÍTULO 5

Casos prácticos de creación de paquetes

Este capítulo ofrece casos prácticos para mostrar situaciones de creación de paquetes como la instalación de objetos de forma condicional, la especificación en el tiempo de ejecución de cuántos archivos se deben crear y la modificación de un archivo de datos durante la instalación y la eliminación de paquetes.

Cada caso práctico comienza por una descripción, seguida de una lista de las técnicas de creación de paquetes utilizadas, una descripción narrativa de la aproximación utilizada a la hora de usar esas técnicas, así como secuencias de comandos y archivos de ejemplo asociados al caso práctico.

A continuación puede ver una lista de los casos prácticos de este capítulo:

Cómo solicitar entrada de información al administrador

El paquete de este caso práctico tiene tres tipos de objetos. El administrador puede elegir cuál de los tres tipos se debe instalar y dónde ubicar los objetos en la máquina de instalación.

Técnicas

Este caso práctico muestra las técnicas siguientes:

- El uso de nombres de ruta paramétricos (variables en nombres de ruta de objetos) que se usan para establecer varios directorios base.

  Para obtener información sobre nombres de ruta paramétricos, consulte "Nombres de rutas paramétricos" en la página 35.
- Uso de una secuencia de comandos `request` para solicitar entrada de información del administrador.

  Para obtener información sobre secuencias de comandos `request`, consulte "Escritura de una secuencia de comandos `request`" en la página 64.
- Configuración de valores condicionales para un parámetro de instalación.

Aproximación

Para configurar la instalación selectiva en este caso práctico, debe completar las tareas siguientes:

- Defina una clase para cada tipo de objeto que se pueda instalar.

  En este caso práctico, los tres tipos de objetos son paquetes ejecutables, páginas de comando man y ejecutables `emacs`. Cada tipo tiene su propia clase: `bin`, `man` y `emacs`, respectivamente. Observe que en el archivo `prototype` todos los archivos de objetos pertenecen a una de estas tres clases.
- Inicialice el parámetro `CLASSES` en el archivo `pkginfo` con el valor de nulo.

  Normalmente, cuando define una clase, debe enumerar esa clase del parámetro `CLASSES` en el archivo `pkginfo`. De lo contrario, no se instalan objetos en esa clase. Para este caso práctico, el parámetro se configura inicialmente con el valor nulo, lo cual significa que no se instalarán objetos. La secuencia de comandos `request` cambiará el parámetro `CLASSES`, de acuerdo con las opciones del administrador. De este modo, el parámetro `CLASSES` se configura sólo en estos tipos de objetos que el administrador desea instalar.

Nota – Por lo general, es una buena idea configurar los parámetros con un valor predeterminado. Si este paquete tuviese componentes comunes a los tres tipos de objetos, podría asignarlos a la clase `none`, y después configurar el parámetro `CLASSES` igual a `none`.

- Inserte nombres de ruta paramétricos en el archivo `prototype`.

  La secuencia de comandos `request` configura estas variables de entorno con el valor que el administrador proporciona. Posteriormente, el comando `pkgadd` resuelve estas variables de entorno en el tiempo de la instalación y sabe dónde instalar el paquete.

  Las tres variables de entorno usadas en este ejemplo se configuran con el valor predeterminado en el archivo `pkginfo` y sirven a los fines siguientes:

  - `$NCMPBIN` define la ubicación de los ejecutables de objetos
  - `$NCMPMAN` define la ubicación de las páginas de comando man
  - `$EMACS` define la ubicación de los ejecutables `emacs`

  El archivo `prototype` de ejemplo muestra cómo definir los nombres de ruta de objeto con variables.

- Cree una secuencia de comandos `request` para preguntar al administrador qué partes del paquete se deben instalar y dónde se deben colocar.

  La secuencia de comandos `request` para este paquete hace al administrador dos preguntas:

  - ¿Se debe instalar esta parte del paquete?

    Si la respuesta es sí, el nombre de clase adecuado se agrega al parámetro `CLASSES`. Por ejemplo, si el administrador elige instalar las páginas de comando man asociadas a este paquete, la clase `man` se agrega al parámetro `CLASSES`.

  - Si es así, ¿dónde se debe colocar esta parte del paquete?

    La variable de entorno adecuada se configura con la respuesta a esta pregunta. En el ejemplo de la página de comando man, la variable `$NCMPMAN` se configura con el valor de la respuesta.

  Estas dos preguntas se repiten para cada uno de los tres tipos de objetos.

  Al final de la secuencia de comandos `request`, los parámetros quedan disponibles para el entorno de instalación para el comando `pkgadd` y otras secuencias de comandos de creación de paquetes. Para ello, la secuencia de comandos `request` escribe estas definiciones en el archivo proporcionado por la utilidad de llamadas. Para este caso práctico no se proporcionan otras secuencias de comandos.

  Al consultar la secuencia de comandos `request` para este caso práctico, tenga en cuenta que las preguntas se generan mediante las herramientas de validación de datos `ckyorn` y `ckpath`. Para obtener más información sobre estas herramientas, consulte `ckyorn(1)` y `ckpath(1)`.

Archivos de casos prácticos

El archivo `pkginfo`

```
PKG=ncmp
NAME=NCMP Utilities
CATEGORY=application, tools
BASEDIR=/
ARCH=SPARC
VERSION=RELEASE 1.0, Issue 1.0
CLASSES=""
NCMPBIN=/bin
NCMPMAN=/usr/man
EMACS=/usr/emacs
```

El archivo `prototype`

```
i pkginfo
i request
x bin $NCMPBIN 0755 root other
f bin $NCMPBIN/dired=/usr/ncmp/bin/dired 0755 root other
f bin $NCMPBIN/less=/usr/ncmp/bin/less 0755 root other
f bin $NCMPBIN/ttype=/usr/ncmp/bin/ttype 0755 root other
f emacs $NCMPBIN/emacs=/usr/ncmp/bin/emacs 0755 root other
x emacs $EMACS 0755 root other
f emacs $EMACS/ansii=/usr/ncmp/lib/emacs/macros/ansii 0644 root other
f emacs $EMACS/box=/usr/ncmp/lib/emacs/macros/box 0644 root other
f emacs $EMACS/crypt=/usr/ncmp/lib/emacs/macros/crypt 0644 root other
f emacs $EMACS/draw=/usr/ncmp/lib/emacs/macros/draw 0644 root other
f emacs $EMACS/mail=/usr/ncmp/lib/emacs/macros/mail 0644 root other
f emacs $NCMPMAN/man1/emacs.1=/usr/ncmp/man/man1/emacs.1 0644 root other
d man $NCMPMAN 0755 root other
d man $NCMPMAN/man1 0755 root other
f man $NCMPMAN/man1/dired.1=/usr/ncmp/man/man1/dired.1 0644 root other
f man $NCMPMAN/man1/ttype.1=/usr/ncmp/man/man1/ttype.1 0644 root other
f man $NCMPMAN/man1/less.1=/usr/ncmp/man/man1/less.1 0644 inixmr other
```

La secuencia de comandos `request`

```
trap 'exit 3' 15
# determine if and where general executables should be placed
ans=`ckyorn -d y \
-p "Should executables included in this package be installed"
` || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES bin"
   NCMPBIN=`ckpath -d /usr/ncmp/bin -aoy \
   -p "Where should executables be installed"
   ` || exit $?
fi
# determine if emacs editor should be installed, and if it should
# where should the associated macros be placed
ans=`ckyorn -d y \
-p "Should emacs editor included in this package be installed"
```

```
‘ || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES emacs"
   EMACS=‘ckpath -d /usr/ncmp/lib/emacs -aoy \
   -p "Where should emacs macros be installed"
   ‘ || exit $?
fi
```

Tenga en cuenta que una secuencia de comandos `request` puede salir sin dejar archivos en el sistema de archivos. En el caso de instalaciones en versiones de Oracle Solaris anteriores a 2.5 y versiones compatibles (donde no se pueda usar una secuencia de comandos `checkinstall`), la secuencia de comandos `request` es el lugar correcto para probar el sistema de archivos del modo necesario con el fin de garantizar que la instalación será satisfactoria. Cuando la secuencia de comandos `request` se termina con el código 1, la instalación se cerrará correctamente.

Estos archivos de ejemplo muestran el uso de rutas paramétricas para establecer varios directorios base. Sin embargo, el método preferido implica el uso del parámetro `BASEDIR` que se administra y se valida mediante el comando `pkgadd`. Siempre que se usen varios directorios base, tenga un cuidado especial en proporcionarlos para la instalación de varias versiones y arquitecturas de la misma plataforma.

Creación de un archivo durante la instalación y cómo guardarlo durante la eliminación

Este caso práctico crea un archivo de base de datos en el tiempo de la instalación y guarda una copia de la base de datos cuando el paquete se suprime.

Técnicas

Este caso práctico muestra las técnicas siguientes:

- Uso de clases y secuencias de comandos de acción de clase para llevar a cabo acciones especiales en diferentes conjuntos de objetos.

  Para obtener más información, consulte “Escritura de secuencias de comandos de acción de clase” en la página 71.

- El uso del archivo `space` para informar al comando `pkgadd` que se necesita espacio extra para instalar este paquete adecuadamente.

  Para obtener más información sobre el archivo `space`, consulte “Reserva del espacio adicional en un sistema de destino” en la página 57.

- Uso del comando `installf` para instalar un archivo que no se ha definido en los archivos `prototype` y `pkgmap`.

Aproximación

Para crear un archivo de base de datos en la instalación y guardar una copia tras la eliminación de este caso práctico, debe completar las tareas siguientes:

- Defina tres clases.

  El paquete de este caso práctico precisa que las tres clases siguientes se definan en el parámetro `CLASSES`:

  - La clase estándar de `none`, que contiene un conjunto de procesos que pertenecen al subdirectorio `bin`.
  - La clase `admin`, que contiene un archivo ejecutable `config` y un directorio que contiene archivos de datos.
  - La clase `cfgdata`, que contiene un directorio.
- Convierta al paquete en reubicable colectivamente.

  Observe que en el archivo `prototype` ninguno de los nombres de ruta comienza por una barra inclinada o una variable de entorno. Esto indica que son reubicables colectivamente.
- Calcule la cantidad de espacio que el archivo de base de datos necesita y cree un archivo `space` que entregar con el paquete. Este archivo avisa al comando `pkgadd` que el paquete necesita espacio extra y especifica cuánto.
- Cree una secuencia de comandos de acción de clase para la clase `admin` (`i.admin`).

  La secuencia de comandos de ejemplo inicializa una base de datos mediante archivos de datos que pertenecen a la clase `admin`. Para llevar a cabo esta tarea, hace lo siguiente:

  - Copia del archivo de datos de origen en el destino adecuado
  - Crea un archivo vacío con el nombre `config.data` y lo asigna a una clase de `cfgdata`
  - Ejecuta el comando `bin/config` (entregado con el paquete y ya instalado) para rellenar el archivo de base de datos `config.data` mediante los archivos de datos que pertenecen a la clase `admin`
  - Ejecuta el comando `installf -f` para terminar la instalación de `config.data`

  No se precisa una acción especial para la clase `admin` en el tiempo de la eliminación por lo que no se crea una secuencia de comandos de acción de clase de eliminación. Esto significa que todos los archivos y directorios de la clase `admin` se suprimen del sistema.
- Cree una secuencia de comandos de acción de clase de eliminación para la clase `cfgdata` (`r.cfgdata`).

  La secuencia de comandos de eliminación hace una copia del archivo de la base de datos antes de que se suprima. No se precisa una acción especial para esta clase en el tiempo de la instalación, por lo que no se necesita una secuencia de comandos de acción de clase de instalación.

Recuerde que la entrada de información a una secuencia de comandos de eliminación es una lista de nombres de ruta que suprimir. Los nombres de ruta siempre aparecen en orden alfabético inverso. Esta secuencia de comandos de eliminación copia archivos en el directorio llamado `$PKGSAV`. Cuando todos los nombres de ruta se han procesado, la secuencia de comandos vuelve y suprime todos los directorios y archivos asociados a la clase `cfgdata`.

La salida de esta secuencia de comandos de eliminación es copiar `config.data` en `$PKGSAV` y después suprimir el archivo `config.data` y el directorio de datos.

Archivos de casos prácticos

El archivo `pkginfo`

```
PKG=krazy
NAME=KrAzY Applications
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1
CLASSES=none cfgdata admin
```

El archivo `prototype`

```
i pkginfo
i request
i i.admin
i r.cfgdata
d none bin 555 root sys
f none bin/process1 555 root other
f none bin/process2 555 root other
f none bin/process3 555 root other
f admin bin/config 500 root sys
d admin cfg 555 root sys
f admin cfg/datafile1 444 root sys
f admin cfg/datafile2 444 root sys
f admin cfg/datafile3 444 root sys
f admin cfg/datafile4 444 root sys
d cfgdata data 555 root sys
```

El archivo `space`

```
# extra space required by config data which is
# dynamically loaded onto the system
data 500 1
```

La secuencia de comandos de acción de clase i.admin

```
# PKGINST parameter provided by installation service
# BASEDIR parameter provided by installation service
while read src dest
```

```
do
   cp $src $dest || exit 2
done
# if this is the last time this script will be executed
# during the installation, do additional processing here.
if [ "$1" = ENDOFCLASS ]
then
# our config process will create a data file based on any changes
# made by installing files in this class; make sure the data file
# is in class ‘cfgdata’ so special rules can apply to it during
# package removal.
   installf -c cfgdata $PKGINST $BASEDIR/data/config.data f 444 root
   sys || exit 2
   $BASEDIR/bin/config > $BASEDIR/data/config.data || exit 2
   installf -f -c cfgdata $PKGINST || exit 2
fi
exit 0
```

Esto ilustra una instancia poco común en la que `installf` es adecuado en una secuencia de comandos de acción de clase. Debido a que se ha usado un archivo `space` para reservar espacio en un sistema de archivos específico, este nuevo archivo se puede agregar de forma segura aunque no se incluya en el archivo `pkgmap`.

La secuencia de comandos de eliminación r.cfgdata

```
# the product manager for this package has suggested that
# the configuration data is so valuable that it should be
# backed up to $PKGSAV before it is removed!
while read path
do
# path names appear in reverse lexical order.
   mv $path $PKGSAV || exit 2
   rm -f $path || exit 2
done
exit 0
```

Definición de dependencias y compatibilidades de paquetes

El paquete de este caso práctico usa archivos de información optativos para definir dependencias y compatibilidades de paquetes, así como para presentar un mensaje de copyright durante la instalación.

Técnicas

Este caso práctico muestra las técnicas siguientes:

- Uso del archivo `copyright`
- Uso del archivo `compver`
- Uso del archivo `depend`

Para obtener más información sobre estos archivos, consulte "Creación de archivos de información" en la página 52.

Aproximación

Para cumplir los requisitos de la descripción debe:

- Crear un archivo `copyright`.

  Un archivo `copyright` contiene el texto ASCII de un mensaje de copyright. El mensaje que se muestra en el archivo de ejemplo aparece en la pantalla durante la instalación del paquete.
- Crear un archivo `compver`.

  El archivo `pkginfo` que aparece en la figura siguiente define la versión de este paquete como versión 3.0. El archivo `compver` define la versión 3.0 como compatible con las versiones 2.3, 2.2, 2.1, 2.1.1, 2.1.3 y 1.7.
- Crear un archivo `depend`.

  Los archivos enumerados en un archivo `depend` ya deben estar instalados en el sistema cuando se instala un paquete. El archivo de ejemplo tiene 11 paquetes que ya deben estar en el sistema en el tiempo de la instalación.

Archivos de casos prácticos

El archivo `pkginfo`

```
PKG=case3
NAME=Case Study #3
CATEGORY=application
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 3.0
CLASSES=none
```

El archivo `copyright`

```
Copyright (c) 1999 company_name
All Rights Reserved.
THIS PACKAGE CONTAINS UNPUBLISHED PROPRIETARY SOURCE CODE OF
company_name.
The copyright notice above does not evidence any
actual or intended publication of such source code
```

El archivo `compver`

```
Version 3.0
Version 2.3
Version 2.2
```

```
Version 2.1
Version 2.1.1
Version 2.1.3
Version 1.7
```

El archivo depend

```
P acu Advanced C Utilities
Issue 4 Version 1
P cc C Programming Language
Issue 4 Version 1
P dfm Directory and File Management Utilities
P ed Editing Utilities
P esg Extended Software Generation Utilities
Issue 4 Version 1
P graph Graphics Utilities
P rfs Remote File Sharing Utilities
Issue 1 Version 1
P rx Remote Execution Utilities
P sgs Software Generation Utilities
Issue 4 Version 1
P shell Shell Programming Utilities
P sys System Header Files
Release 3.1
```

Modificación de un archivo mediante secuencias de comandos de acción de clase y clases estándar

Este caso práctico modifica un archivo durante la instalación del paquete mediante secuencias de comandos de acción de clase y clases estándar. Usa uno de los tres métodos de modificación. Los otros dos métodos se describen en "Modificación de un archivo mediante la clase `sed` y una secuencia de comandos `postinstall`" en la página 115 y "Modificación de un archivo mediante la clase `build`" en la página 117. El archivo modificado es `/etc/inittab` .

Técnicas

Este caso práctico muestra cómo usar secuencias de comandos de acción de clase de instalación y eliminación. Para obtener más información, consulte "Escritura de secuencias de comandos de acción de clase" en la página 71.

Aproximación

Para modificar `/etc/inittab` durante la instalación, mediante secuencias de comandos de acción de clase y clases, debe completar las tareas siguientes:

- Crear una clase.

Cree una clase llamada `inittab` . Debe proporcionar una secuencia de comandos de acción de clase de instalación y eliminación para esta clase. Defina la clase `inittab` en el parámetro `CLASSES` del archivo `pkginfo`.

- Crear un archivo `inittab`.

  Este archivo contiene la información para la entrada que desea agregar a `/etc/inittab` . Observe en la figura del archivo `prototype` que `inittab` forma parte de la clase `inittab` y que tiene un tipo de archivo de `e` para editar.

- Crear una secuencia de comandos de acción de clase de instalación (`i.inittab`).

  Recuerde que las secuencias de comandos de acción de clase deben producir los mismos resultados cada vez que se ejecutan. La secuencia de comandos de acción de clase ejecuta estos pasos:

  - Comprueba si la entrada se ha agregado anteriormente
  - Si es así, suprime las versiones anteriores de la entrada
  - Modifica el archivo `inittab` y agrega líneas de comentarios para que sepa de dónde procede la entrada
  - Mueve el archivo temporal a `/etc/inittab`
  - Ejecuta el comando `init q` cuando recibe el indicador `ENDOFCLASS`

  Tenga en cuenta que el comando `init q` se puede ejecutar mediante esta secuencia de comandos de instalación. Esta aproximación no necesita una secuencia de comandos de `postinstall` de una línea.

- Crear una secuencia de comandos de acción de clase de eliminación (`r.inittab`).

  La secuencia de comandos de eliminación es muy parecida a la de instalación. La información agregada por la secuencia de comandos de instalación se suprime y se ejecuta el comando `init q`.

Este caso práctico es más complicado que el siguiente; consulte "Modificación de un archivo mediante la clase `sed` y una secuencia de comandos `postinstall`" en la página 115. En lugar de proporcionar dos archivos se necesitan tres; el archivo `/etc/inittab` entregado es simplemente un marcador de posición que contiene un fragmento de la entrada que se va a insertar. Ésta se podría haber colocado en el archivo `i.inittab`, a excepción de que el comando `pkgadd` debe tener un archivo para pasar al archivo `i.inittab`. Asimismo, el procedimiento de eliminación debe colocarse en un archivo aparte (`r.inittab`). Mientras este método funcione bien, es mejor reservarlo para casos que impliquen instalaciones muy complicadas de varios archivos. Consulte "Modificación de archivos `crontab` durante la instalación" en la página 119.

El programa `sed` usado en "Modificación de un archivo mediante la clase `sed` y una secuencia de comandos `postinstall`" en la página 115 admite varias instancias de paquetes, ya que el comentario del final de la entrada `inittab` se basa en la instancia del paquete. El caso práctico de "Modificación de un archivo mediante la clase `build`" en la página 117 muestra una aproximación más simplificada para la modificación de `/etc/inittab` durante la instalación.

Archivos de casos prácticos

El archivo `pkginfo`

```
PKG=case5
NAME=Case Study #5
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=inittab
```

El archivo `prototype`

```
i pkginfo
i i.inittab
i r.inittab
e inittab /etc/inittab ? ? ?
```

La secuencia de comandos de acción de clase de instalación i.inittab

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
exit 2
sed -e "s/$/#$PKGINST" $src >> /tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
if [ "$1" = ENDOFCLASS ]
then
/sbin/init q ||
exit 2
fi
exit 0
```

La secuencia de comandos de acción de clase de eliminación r.inittab

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# are associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
/sbin/init q ||
```

```
exit 2
exit 0
```

El archivo inittab

```
rb:023456:wait:/usr/robot/bin/setup
```

Modificación de un archivo mediante la clase sed y una secuencia de comandos postinstall

Este caso práctico modifica un archivo que existe en la máquina de instalación durante la instalación del paquete. Usa uno de los tres métodos de modificación. Los otros dos métodos se describen en "Modificación de un archivo mediante secuencias de comandos de acción de clase y clases estándar" en la página 112 y "Modificación de un archivo mediante la clase build" en la página 117. El archivo modificado es /etc/inittab.

Técnicas

Este caso práctico muestra las técnicas siguientes:

- Uso de la clase sed

  Para obtener más información sobre la clase sed, consulte "Secuencia de comandos de clase sed" en la página 76.

- Uso de una secuencia de comandos postinstall

  Para obtener más información sobre esta secuencia de comandos, consulte "Escritura de secuencias de comandos de procedimientos" en la página 69.

Aproximación

Para modificar /etc/inittab en el tiempo de la instalación mediante la clase sed, debe completar las tareas siguientes:

- Agregue la secuencia de comandos de clase sed al archivo prototype.

  El nombre de una secuencia de comandos debe ser el mismo que el del archivo que se modificará. En este caso, el archivo que se debe modificar es /etc/inittab y por lo tanto la secuencia de comandos sed debe recibir el nombre de /etc/inittab. No hay requisitos para el modo, propietario y grupo de una secuencia de comandos sed (se representa en el ejemplo de prototype mediante signos de interrogación). El tipo de archivo de la secuencia de comandos sed debe ser e (para indicar que es modificable).

- Configure el parámetro CLASSES para que incluya la clase sed.

  Tal como se muestra en el archivo de ejemplo, sed es la única clase que se instala. Sin embargo, puede ser cualquier número de clases.

- Crear una secuencia de comandos de acción de clase sed.

  El paquete no puede entregar una copia de /etc/inittab que tenga el aspecto que necesite, ya que /etc/inittab es un archivo dinámico y no tiene forma de saber el aspecto que tendrá en el tiempo de la instalación del paquete. Sin embargo, el uso de una secuencia de comandos sed permite modificar el archivo /etc/inittab durante la instalación del paquete.

- Crear una secuencia de comandos postinstall.

  Debe ejecutar el comando init q para informar al sistema de que /etc/inittab se ha modificado. El único lugar en que puede ejecutar dicha acción en este ejemplo es en una secuencia de comandos postinstall. Si mira a la secuencia de comandos postinstall de ejemplo, verá que su única finalidad es ejecutar el comando init q.

Esta aproximación a la modificación de /etc/inittab durante la instalación tiene un inconveniente: debe entregar una secuencia de comandos completa (la secuencia de comandos postinstall) simplemente para ejecutar el comando init q.

Archivos de casos prácticos

El archivo pkginfo

```
PKG=case4
NAME=Case Study #4
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=sed
```

El archivo prototype

```
i pkginfo
i postinstall
e sed /etc/inittab ? ? ?
```

La secuencia de comandos de acción de clase sed Class Action Script (/etc/inittab)

```
!remove
# remove all entries from the table that are associated
# with this package, though not necessarily just
# with this package instance
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
!install
# remove any previous entry added to the table
# for this particular change
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
# add the needed entry at the end of the table;
```

```
# sed(1) does not properly interpret the '$a'
# construct if you previously deleted the last
# line, so the command
# $a\
# rb:023456:wait:/usr/robot/bin/setup #ROBOT
# will not work here if the file already contained
# the modification. Instead, you will settle for
# inserting the entry before the last line!
$i\
rb:023456:wait:/usr/robot/bin/setup #ROBOT
```

La secuencia de comandos `postinstall`

```
# make init re-read inittab
/sbin/init q ||
exit 2
exit 0
```

Modificación de un archivo mediante la clase build

Este caso práctico modifica un archivo que existe en la máquina de instalación durante la instalación del paquete. Usa uno de los tres métodos de modificación. Los otros dos métodos se describen en "Modificación de un archivo mediante secuencias de comandos de acción de clase y clases estándar" en la página 112 y "Modificación de un archivo mediante la clase `sed` y una secuencia de comandos `postinstall`" en la página 115. El archivo modificado es `/etc/inittab`.

Técnicas

En este caso práctico se muestra cómo usar la clase `build`. Para obtener más información sobre la clase `build`, consulte "Secuencia de comandos de clase `build`" en la página 77.

Aproximación

Esta aproximación a la modificación de `/etc/inittab` usa la clase `build`. Una secuencia de comandos de clase `build` se ejecuta como secuencia de comandos de shell y su salida se convierte en la nueva versión del archivo que se está ejecutando. En otras palabras, el archivo de datos `/etc/inittab` que se entrega con este paquete se ejecutará y la salida de dicha ejecución se convertirá en `/etc/inittab`.

La secuencia de comandos de clase `build` se ejecuta durante la instalación y la eliminación del paquete. El argumento `install` se pasa al archivo si se ejecuta en el tiempo de la instalación. Observe en la secuencia de comandos de clase `build` de ejemplo que las acciones de instalación se definen mediante la prueba para este argumento.

Para editar `/etc/inittab` mediante la clase `build`, debe completar las tareas siguientes:

- Definir el archivo build en el archivo `prototype`.

  La entrada para el archivo build en el archivo `prototype` debe colocarla en la clase `build` y definir su tipo de archivo como `e`. Asegúrese de que el parámetro `CLASSES` del archivo `pkginfo` se define como `build`.
- Crear la secuencia de comandos de clase `build`.

  La secuencia de comandos de clase `build` de ejemplo ejecuta los procesos siguientes:

  - Edita el archivo `/etc/inittab` para suprimir los cambios en este paquete. Observe que el nombre de archivo `/etc/inittab` no es modificable en el comando `sed`.
  - Si se está instalando el paquete, agrega la nueva línea al final de `/etc/inittab`. Se incluye una etiqueta de comentario en esta nueva entrada para describir de dónde viene dicha entrada.
  - Ejecuta el comando `init q`.

Esta solución aborda los inconvenientes descritos en los casos prácticos de "Modificación de un archivo mediante secuencias de comandos de acción de clase y clases estándar" en la página 112 y "Modificación de un archivo mediante la clase `sed` y una secuencia de comandos `postinstall`" en la página 115. Sólo se necesita un archivo corto (más allá de los archivos `pkginfo` y `prototype`). El archivo funciona con varias instancias de un paquete debido a que se usa el parámetro `PKGINST` y no se requiere una secuencia de comandos `postinstall`, ya que el comando `init q` se puede ejecutar desde la secuencia de comandos de clase `build`.

Archivos de casos prácticos

El archivo `pkginfo`

```
PKG=case6
NAME=Case Study #6
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=build
```

El archivo `prototype`

```
i pkginfo
e build /etc/inittab ? ? ?
```

El archivo build

```
# PKGINST parameter provided by installation service
# remove all entries from the existing table that
# are associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" /etc/inittab ||
exit 2
```

```
if [ "$1" = install ]
then
# add the following entry to the table
echo "rb:023456:wait:/usr/robot/bin/setup #$PKGINST" ||
exit 2
fi
/sbin/init q ||
exit 2
exit 0
```

Modificación de archivos crontab durante la instalación

Este caso práctico modifica los archivos crontab durante la instalación del paquete.

Técnicas

Este caso práctico muestra las técnicas siguientes:

- Uso de secuencias de comandos de acción de clase y clases

  Para obtener más información, consulte "Escritura de secuencias de comandos de acción de clase" en la página 71.
- Uso del comando crontab en una secuencia de comandos de acción de clase.

Aproximación

El modo más eficaz de modificar más de un archivo durante la instalación es definir una clase y ofrecer una secuencia de comandos de acción de clase. Si se ha servido de la aproximación a la clase build, debe entregar una secuencia de comandos de clase build para cada archivo crontab modificado. La definición de una clase cron proporciona una aproximación más general. Para editar los archivos crontab con esta aproximación, debe:

- Definir los archivos crontab que se van a modificar en el archivo prototype.

  Cree una entrada en el archivo prototype para cada archivo crontab que se modificará. Defina la clase como cron y el tipo de archivo como e para cada archivo. Use el nombre real del archivo que se va a modificar.
- Crear los archivos crontab para el paquete.

  Estos archivos contienen la información que desea agregar a los archivos crontab existentes del mismo nombre.
- Crear una secuencia de comandos de acción de clase de instalación para la clase cron.

  La secuencia de comandos i.cron de ejemplo ejecuta los procesos siguientes:

  - Determina el ID de usuario (UID).

La secuencia de comandos i.cron configura la variable *user* con el nombre base de la secuencia de comandos de clase cron que se está procesando. Ese nombre es el UID. Por ejemplo, el nombre base de /var/spool/cron/crontabs/root es raíz, además del UID.

- Ejecuta crontab mediante el UID y la opción -l.

  El uso de la opción -l indica a crontab que envíe el contenido del archivo crontab del usuario definido a la salida estándar.
- Dirige la salida del comando crontab a una secuencia de comandos sed que suprime las entradas anteriores agregadas con esta técnica de instalación.
- Coloca la salida modificada en un archivo temporal.
- Agrega el archivo de datos del UID raíz (que se entregó con el paquete) al archivo temporal y agrega una etiqueta para que sepa de dónde vienen estas entradas.
- Ejecuta crontab con el mismo UID y le entrega el archivo temporal como entrada de información.

- Crear una secuencia de comandos de acción de clase de eliminación para la clase cron.

  La secuencia de comandos r.cron es la misma que la secuencia de comandos de instalación excepto que no hay procesos para agregar información al archivo crontab.

  Estos procesos se llevan a cabo para cada archivo en la clase cron.

Archivos de casos prácticos

Un superusuario ejecuta las secuencias de comandos i.cron y r.cron que se describen a continuación. La modificación del archivo crontab de otro usuario puede tener resultados imprevistos. Si fuera necesario, cambie la entrada siguiente de cada secuencia de comandos:

```
crontab $user < /tmp/$$crontab ||
```

a

```
su $user -c "crontab /tmp/$$crontab" ||
```

El comando pkginfo

```
PKG=case7
NAME=Case Study #7
CATEGORY=application
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1.0
CLASSES=cron
```

El archivo prototype

```
i pkginfo
i i.cron
```

```
i r.cron
e cron /var/spool/cron/crontabs/root ? ? ?
e cron /var/spool/cron/crontabs/sys ? ? ?
```

La secuencia de comandos de acción de clase de instalación i.cron

```
# PKGINST parameter provided by installation service
while read src dest
do
user=‘basename $dest‘ ||
exit 2
(crontab -l $user |
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
sed -e "s/$/#$PKGINST/" $src >> /tmp/$$crontab ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
rm -f /tmp/$$crontab
done
exit 0
```

La secuencia de comandos de acción de clase de eliminación r.cron

```
# PKGINST parameter provided by installation service
while read path
do
user=‘basename $path‘ ||
exit 2
(crontab -l $user |
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
rm -f /tmp/$$crontab
done
exit
```

Archivo `crontab` n.º 1

```
41,1,21 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /usr/bin/su uucp -c
"/usr/lib/uucp/uudemon.cleanup" >
/dev/null 2>&1
11,31,51 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
```

Archivo `crontab` n.º 2

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

Nota – Si la modificación de un grupo de archivos aumenta el tamaño total del archivo en más de 10K, proporcione un archivo `space` para que el comando `pkgadd` pueda permitir este aumento. Para obtener más información sobre el archivo `space`, consulte "Reserva del espacio adicional en un sistema de destino" en la página 57.

Instalación y eliminación de un controlador con secuencias de comandos de procedimientos

Este paquete instala un controlador.

Técnicas

Este caso práctico muestra las técnicas siguientes:

- Instalación y carga de un controlador con una secuencia de comandos `postinstall`
- Descarga de un controlador con una secuencia de comandos `preremove`

Para obtener más información sobre estas secuencias de comandos, consulte "Escritura de secuencias de comandos de procedimientos" en la página 69.

Aproximación

- Crear una secuencia de comandos `request`.

  La secuencia de comandos `request` determina dónde el administrador desea que se instalen los objetos del controlador: pregunta al administrador y asigna la respuesta al parámetro `$KERNDIR`.

  La secuencia de comandos termina con una rutina para hacer que los dos parámetros `CLASSES` y `KERNDIR` estén disponibles en el entorno de instalación y la secuencia de comandos `postinstall`.

- Crear una secuencia de comandos `postinstall`.

  La secuencia de comandos `postinstall` lleva a cabo, de hecho, la instalación del controlador. Se ejecuta después de que los dos archivos `buffer` y `buffer.conf` se hayan instalado. El archivo `postinstall` mostrado en este ejemplo lleva a cabo las acciones siguientes:

  - Usa el comando `add_drv` para cargar el controlador en el sistema.
  - Crea un vínculo para el dispositivo mediante el comando `installf`.
  - Termina la instalación mediante el comando `installf -f`.

- Crea una secuencia de comandos `preremove`.

  La secuencia de comandos `preremove` usa el comando `rem_drv` para descargar el controlador del sistema y después suprime el vínculo `/dev/buffer0` .

Archivos de casos prácticos

El archivo `pkginfo`

```
PKG=bufdev
NAME=Buffer Device
CATEGORY=system
BASEDIR=/
ARCH=INTEL
VERSION=Software Issue #19
CLASSES=none
```

El archivo `prototype`

Para instalar un controlador en el tiempo de la instalación, debe incluir los archivos de configuración y objetos para el controlador en el archivo `prototype`.

En este ejemplo, el módulo ejecutable del controlador recibe el nombre de `buffer`; el comando `add_drv` funciona en este archivo. El núcleo usa el archivo de configuración, `buffer.conf`, para ayudar a configurar el controlador.

```
i pkginfo
i request
i postinstall
i preremove
f none $KERNDIR/buffer 444 root root
f none $KERNDIR/buffer.conf 444 root root
```

Si mira al archivo `prototype` de este ejemplo, puede observar lo siguiente:

- Puesto que no se necesita un tratamiento especial para los objetos de paquete, puede colocarlos en la clase `none` estándar. El parámetro `CLASSES` se configura con el valor `none` en el archivo `pkginfo`.
- Los nombres de ruta de `buffer` y `buffer.conf` comienzan por la variable `$KERNDIR`. Esta variable se configura en la secuencia de comandos `request` y permite al administrador decidir dónde se deben instalar los archivos del controlador. El directorio predeterminado es `/kernel/drv`.
- Hay una entrada para la secuencia de comandos `postinstall` (la secuencia de comandos que ejecutará la instalación del controlador).

La secuencia de comandos `request`

```
trap 'exit 3' 15
# determine where driver object should be placed; location
# must be an absolute path name that is an existing directory
KERNDIR=`ckpath -aoy -d /kernel/drv -p \
"Where do you want the driver object installed"` || exit $?

# make parameters available to installation service, and
# so to any other packaging scripts
cat >$1 <<!

CLASSES='$CLASSES'
KERNDIR='$KERNDIR'
!
exit 0
```

La secuencia de comandos `postinstall`

```
# KERNDIR parameter provided by `request' script
err_code=1                    # an error is considered fatal
# Load the module into the system
cd $KERNDIR
add_drv -m '* 0666 root sys' buffer || exit $err_code
# Create a /dev entry for the character node
installf $PKGINST /dev/buffer0=/devices/eisa/buffer*:0 s
installf -f $PKGINST
```

La secuencia de comandos `preremove`

```
err_code=1                    # an error is considered fatal
# Unload the driver
rem_drv buffer || exit $err_code
# remove /dev file
removef $PKGINST /dev/buffer0 ; rm /dev/buffer0
removef -f $PKGINST
```

Instalación de un controlador mediante las secuencias de comandos de procedimientos y la clase sed

En este caso práctico se describe cómo instalar un controlador mediante las secuencias de comandos de procedimientos y la clase `sed`. También es diferente del caso práctico anterior (consulte "Instalación y eliminación de un controlador con secuencias de comandos de procedimientos" en la página 122) porque este paquete se compone de objetos absolutos y reubicables.

Técnicas

Este caso práctico muestra las técnicas siguientes:

- Construcción de un archivo `prototype` con objetos reubicables y absolutos.

Para obtener más información sobre la construcción de un archivo `prototype`, consulte "Creación de un archivo `prototype`" en la página 31.

- Uso de una secuencia de comandos `postinstall`

  Para obtener más información sobre esta secuencia de comandos, consulte "Escritura de secuencias de comandos de procedimientos" en la página 69.

- Uso de una secuencia de comandos `preremove`.

  Para obtener más información sobre esta secuencia de comandos, consulte "Escritura de secuencias de comandos de procedimientos" en la página 69.

- Uso de un archivo `copyright`

  Para obtener más información sobre este archivo, consulte "Escritura de un mensaje de copyright" en la página 55.

Aproximación

- Crear un archivo `prototype` que contenga objetos de paquetes reubicables y absolutos.

  Puede ver todos los detalles en "El archivo `prototype`" en la página 126.

- Agregue la secuencia de comandos de clase `sed` al archivo `prototype`.

  El nombre de una secuencia de comandos debe ser el mismo que el del archivo que se modificará. En este caso, el archivo que se debe modificar es `/etc/devlink.tab` y por lo tanto la secuencia de comandos `sed` recibe el nombre de `/etc/devlink.tab`. No hay requisitos para el modo, propietario y grupo de una secuencia de comandos `sed` (se representa en el ejemplo de `prototype` mediante signos de interrogación). El tipo de archivo de la secuencia de comandos `sed` debe ser `e` (para indicar que es modificable).

- Configure el parámetro `CLASSES` para que incluya la clase `sed`.
- Crear una secuencia de comandos de acción de clase `sed` (`/etc/devlink.tab`).
- Crear una secuencia de comandos `postinstall`.

  La secuencia de comandos `postinstall` necesita ejecutar el comando `add_drv` para agregar el controlador de dispositivos al sistema.

- Crear una secuencia de comandos `preremove`.

  La secuencia de comandos `preremove` necesita ejecutar el comando `rem_drv` para suprimir el controlador de dispositivos del sistema, antes del paquete que se va a suprimir.

- Crear un archivo `copyright`.

  Un archivo `copyright` contiene el texto ASCII de un mensaje de copyright. El mensaje que se muestra en el archivo de ejemplo aparece en la pantalla durante la instalación del paquete.

Archivos de casos prácticos

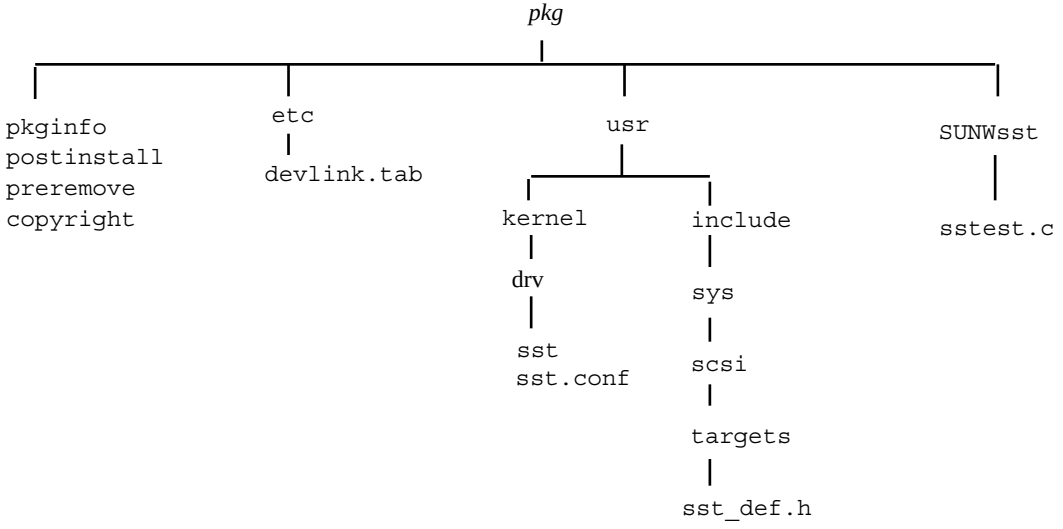
El archivo `pkginfo`

```
PKG=SUNWsst
NAME=Simple SCSI Target Driver
VERSION=1
CATEGORY=system
ARCH=sparc
VENDOR=Sun Microsystems
BASEDIR=/opt
CLASSES=sed
```

El archivo `prototype`

Por ejemplo, este caso práctico usa la distribucion jerárquica de los objetos de paquete que se muestran en la figura siguiente.

FIGURA 5–1 Estructura jerárquica de directorios de paquetes



Los objetos el paquete se instalan en los mismos lugares que en el directorio `pkg` superior. Los módulos del controlador (`sst` y `sst.conf`) se instalan en `/usr/kernel/drv` y el archivo de inclusión se instala en `/usr/include/sys/scsi/targets`. Los archivos `sst`, `sst.conf` y `sst_def.h` son objetos absolutos. El programa de prueba, `sstest.c`, y su directorio `SUNWsst`, son reubicables; su ubicación de instalación se configura mediante el parámetro `BASEDIR`.

Los demás componentes del paquete (todos los archivos de control) se encuentran en el directorio superior del paquete en la máquina de desarrollo, excepto la secuencia de comandos

de clase `sed`. Éste recibe el nombre de `devlink.tab` después del archivo al que modifica, y se sitúa en `etc`, el directorio que contiene el archivo `devlink.tab` real.

En el directorio `pkg`, ejecute el comando `pkgproto` del modo siguiente:

```
find usr SUNWsst -print | pkgproto > prototype
```

La salida del comando anterior tiene este aspecto:

```
d none usr 0775 pms mts
d none usr/include 0775 pms mts
d none usr/include/sys 0775 pms mts
d none usr/include/sys/scsi 0775 pms mts
d none usr/include/sys/scsi/targets 0775 pms mts
f none usr/include/sys/scsi/targets/sst_def.h 0444 pms mts
d none usr/kernel 0775 pms mts
d none usr/kernel/drv 0775 pms mts
f none usr/kernel/drv/sst 0664 pms mts
f none usr/kernel/drv/sst.conf 0444 pms mts
d none SUNWsst 0775 pms mts
f none SUNWsst/sstest.c 0664 pms mts
```

Este archivo `prototype` aún no está completo. Para completar este archivo, necesita hacer las modificaciones siguientes:

- Inserte las entradas para los archivos de control (tipo de archivo `i`), ya que tienen un formato diferente de los demás objetos del paquete.
- Suprima entradas de los directorios que ya existen en el sistema de destino.
- Cambie el permiso de acceso y la propiedad de cada entrada.
- Anteponga una barra oblicua a los objetos de paquetes absolutos.

Éste es el archivo `prototype` final:

```
i pkginfo
i postinstall
i preremove
i copyright
e sed /etc/devlink.tab ? ? ?
f none /usr/include/sys/scsi/targets/sst_def.h 0644 bin bin
f none /usr/kernel/drv/sst 0755 root sys
f none /usr/kernel/drv/sst.conf 0644 root sys
d none SUNWsst 0775 root sys
f none SUNWsst/sstest.c 0664 root sys
```

Los signos de interrogación en la entrada para la secuencia de comandos `sed` indican que los permisos de acceso y la propiedad del archivo existente en la maquina de instalación no se deben cambiar.

La secuencia de comandos de acción de clase sed Class Action Script (/etc/devlink.tab)

En el ejemplo de controlador, se utiliza una secuencia de comandos de clase `sed` con el fin de agregar una entrada para el controlador en el archivo `/etc/devlink.tab`. El comando `devlinks` utiliza este archivo para crear vínculos simbólicos de `/dev` en `/devices`. Ésta es la secuencia de comandos `sed`:

```
# sed class script to modify /etc/devlink.tab
!install
/name=sst;/d
$i\
type=ddi_pseudo;name=sst;minor=character    rsst\\A1

!remove
/name=sst;/d
```

El comando `pkgrm` no ejecuta la parte de eliminación de la secuencia de comandos. Puede que necesite agregar una línea a la secuencia de comandos `preremove` para ejecutar directamente `sed` con el fin de suprimir la entrada del archivo `/etc/devlink.tab`.

La secuencia de comandos de instalación `postinstall`

En este ejemplo, todo lo que necesita hacer la secuencia de comandos es ejecutar el comando `add_drv`.

```
# Postinstallation script for SUNWsst
# This does not apply to a client.
if [$PKG_INSTALL_ROOT = "/" -o -z $PKG_INSTALL_ROOT]; then
   SAVEBASE=$BASEDIR
   BASEDIR=””; export BASEDIR
   /usr/sbin/add_drv sst
   STATUS=$?
   BASEDIR=$SAVEBASE; export BASEDIR
   if [ $STATUS -eq 0 ]
   then
         exit 20
   else
         exit 2
   fi
else
   echo "This cannot be installed onto a client."
   exit 2
fi
```

El comando `add_drv` usa el parámetro `BASEDIR`, de forma que la secuencia de comandos debe anular la configuración de `BASEDIR` antes de ejecutar el comando y restaurarla después.

Una de las acciones del comando `add_drv` es ejecutar `devlinks` que usa la entrada situada en `/etc/devlink.tab` por la secuencia de comandos de clase `sed` para crear las entradas `/dev` para el controlador.

El código de salida de la secuencia de comandos `postinstall` es importante. El código de salida `20` comunica al comando `pkgadd` que indique al usuario que debe reiniciar el sistema (necesario después de instalar un controlador); el código de salida `2` comunica al comando `pkgadd` que indique al usuario que la instalación ha fallado de manera parcial.

La secuencia de comandos de eliminación `preremove`

En el caso de este ejemplo de controlador, suprime los vínculos en `/dev` y ejecuta el comando `rem_drv` en el controlador.

```
# Pre removal script for the sst driver
echo “Removing /dev entries”
/usr/bin/rm -f /dev/rsst*

echo “Deinstalling driver from the kernel”
SAVEBASE=$BASEDIR
BASEDIR=””; export BASEDIR
/usr/sbin/rem_drv sst
BASEDIR=$SAVEBASE; export BASEDIR

exit
```

La secuencia de comandos suprime por sí misma las entradas de `/dev`; el comando `rem_drv` suprime las entradas `/devices`.

El archivo `copyright`

Éste es un archivo ASCII simple que contiene el texto de un aviso de copyright. El aviso se muestra al comienzo de la instalación del paquete, exactamente como aparece en el archivo.

```
    Copyright (c) 1999 Drivers-R-Us, Inc.
    10 Device Drive, Thebus, IO 80586

All rights reserved. This product and related documentation is
protected by copyright and distributed under licenses
restricting its use, copying, distribution and decompilation.
No part of this product or related documentation may be
reproduced in any form by any means without prior written
authorization of Drivers-R-Us and its licensors, if any.
```

◆ ◆ ◆ CAPÍTULO 6

Técnicas avanzadas para la creación de paquetes

Las capacidades completas de empaquetado de System V que se implementan en el sistema operativo Oracle Solaris proporcionan una herramienta efectiva para la instalación de productos de software. Como diseñador de paquetes, puede beneficiarse de estas posibilidades. Los paquetes que no forman parte del sistema operativo Oracle Solaris (paquetes no integrados) pueden usar el mecanismo de clase para personalizar las instalaciones de servidores y clientes. Es posible diseñar paquetes reubicables para satisfacer los deseos del administrador. Se puede entregar un producto complejo como un conjunto de paquetes compuestos que resuelven automáticamente las dependencias de los paquetes. La actualización y la aplicación de parches se puede personalizar mediante el diseñador de paquetes. Los paquetes con parches se pueden entregar del mismo modo que los paquetes sin parches, y los archivos de recuperación también se pueden incluir en el producto.

A continuación, se muestra una lista de la información general de este capítulo:

Especificación del directorio base

Puede usar diversos métodos para especificar dónde se instalará un paquete; es importante que se pueda cambiar la base de instalación dinámicamente en el tiempo de la instalación. Si se lleva a cabo correctamente, un administrador puede instalar varias versiones y varias arquitecturas sin problemas.

En esta sección se tratan en primer lugar los métodos habituales, seguidos de las aproximaciones que mejoran la instalación en sistemas heterogéneos.

El archivo administrativo predeterminado

Los administradores responsables de la instalación de los paquetes pueden usar los archivos de administración para controlar la instalación de los paquetes. Sin embargo, como diseñador de paquetes, debe conocer los archivos de administración y cómo un administrador puede modificar la instalación deseada del paquete.

Un archivo de administración indica al comando `pkgadd` si se debe efectuar alguna de las comprobaciones o solicitudes que normalmente realiza. En consecuencia, los administradores deben comprender completamente el proceso de instalación de un paquete y las secuencias de comandos implicadas antes de usar los archivos de administración.

Se distribuye un archivo administrativo predeterminado básico con el sistema operativo SunOS en `/var/sadm/install/admin/default`. Éste es el archivo que establece el nivel más básico de directiva de administración respecto a la instalación de productos de software. El archivo tiene este aspecto al distribuirlo:

```
#ident "@(#)default
1.4 92/12/23 SMI"    /* SVr4.0 1.5.2.1    */
mail=
instance=unique
partial=ask
runlevel=ask
idepend=ask
rdepend=ask
space=ask
setuid=ask
conflict=ask
action=ask
basedir=default
```

El administrador puede editar este archivo para establecer nuevos comportamientos predeterminados, o bien para crear un archivo de administración diferente y especificar su existencia mediante el uso de la opción `-a` en el comando `pkgadd`.

Es posible definir once parámetros en un archivo de administración, aunque no es necesario definirlos todos. Para obtener más información, consulte `admin(4)`.

El parámetro `basedir` especifica cómo se derivará el directorio base cuando se instale un paquete. La mayoría de los administradores lo dejan como `default`, pero `basedir` se puede establecer con uno de los valores siguientes:

- `ask`: significa que debe pedir siempre al administrador un directorio base
- Un nombre de ruta absoluta
- Un nombre de ruta absoluta que contenga la construcción `$PKGINST`, lo cual significa que debe hacer siempre la instalación en un directorio base derivado de la instancia del paquete

Nota – Si se llama al comando `pkgadd` con el argumento `-a none`, siempre pide al administrador un directorio base. Desafortunadamente, de este modo se configuran *todos* los parámetros del archivo con el valor predeterminado de `quit`, lo cual puede suponer problemas adicionales.

Familiarización con la incertidumbre

Un administrador tiene control sobre todos los paquetes que se instalan en un sistema mediante el uso de un archivo de administración. Desafortunadamente, el *diseñador de paquetes* proporciona a menudo un archivo administrativo alternativo predeterminado, independientemente de los deseos del administrador.

Los diseñadores de paquetes contienen en ocasiones un archivo de administración alternativo, por lo que ellos, no el administrador, controlan la instalación de un paquete. Debido a que la entrada `basedir` del archivo administrativo predeterminado anula los demás directorios base, ofrece un método sencillo para seleccionar el directorio base adecuado en el tiempo de la instalación. En todas las versiones del sistema operativo Oracle Solaris anteriores a la versión Solaris 2.5, se consideraba el método más sencillo para controlar el directorio base.

Sin embargo, es necesario que acepte los deseos del administrador respecto a la instalación del producto. La distribución de un archivo administrativo temporal predeterminado con el fin de controlar la instalación conduce a desconfianza por parte de los administradores. Debe usar las secuencias de comandos `request` y `checkinstall` para controlar estas instalaciones bajo la supervisión del administrador. Si la secuencia de comandos `request` implica fielmente al administrador en el proceso, la creación de paquetes de System V servirá a los diseñadores de paquetes y los administradores.

Uso del parámetro `BASEDIR`

El archivo `pkginfo` para cualquier paquete reubicable debe incluir un directorio base predeterminado en forma de entrada como la siguiente:

`BASEDIR=`*absolute_path*

Éste es únicamente el directorio base predeterminado; el administrador puede cambiarlo durante la instalación.

Mientras algunos paquetes requieren más de un directorio base, la ventaja de usar este parámetro para colocar el paquete se debe a que se garantiza que el directorio base estará en su lugar y será modificable como directorio válido en el momento en el que comience la instalación. La ruta correcta al directorio base para el servidor y el cliente está disponible para todas las secuencias de comandos de procedimientos en forma de variables de entorno reservadas, y el comando `pkginfo -r SUNWstuf` muestra la instalación actual base para el paquete.

En la secuencia de comandos `checkinstall`, `BASEDIR` es el parámetro definido exactamente en el archivo `pkginfo` (aún no se ha acondicionado). Con el fin de inspeccionar el directorio base de destino, se precisa la construcción `${PKG_INSTALL_ROOT}$BASEDIR`. Esto significa que la secuencia de comandos `request` o `checkinstall` puede cambiar el valor de `BASEDIR` en el entorno de instalación con resultados predecibles. En el tiempo de llamar a la secuencia de comandos `preinstall`, el parámetro `BASEDIR` es el puntero completamente acondicionado al directorio base real en el sistema de destino, incluso si el sistema es un cliente.

Nota – La secuencia de comandos `request` utiliza el parámetro `BASEDIR` de forma diferente según las versiones del sistema operativo SunOS. Con el fin de probar un parámetro `BASEDIR` en una secuencia de comandos `request`, se debe usar el código siguiente para determinar el directorio de base real en uso.

```
# request script
constructs base directory
if [ ${CLIENT_BASEDIR} ]; then
      LOCAL_BASE=$BASEDIR
else
      LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
fi
```

Uso de directorios base paramétricos

Si un paquete necesita diversos directorios base, puede establecerlos con nombres de ruta paramétricos. Este método se ha vuelto bastante popular, aunque tiene los siguientes inconvenientes:

- Un paquete con nombres de ruta paramétricos se comporta normalmente como un paquete absoluto pero el comando `pkgadd` lo considera un paquete reubicable. El parámetro `BASEDIR` debe definirse aunque no se use.
- El administrador no puede establecer la base de instalación para el paquete mediante las utilidades de System V (el comando `pkginfo -r` no funcionará).
- El administrador no puede usar el método establecido para reubicar el paquete (se llama reubicable pero actúa de forma absoluta).
- Las instalaciones de varias versiones o arquitecturas necesitan planificar las contingencias para cada uno de los directorios base de destino, lo cual significa a menudo diversas secuencias de comandos complejas de acción de clase.

Mientras los parámetros que determinan los directorios base se definen en el archivo `pkginfo`, se pueden modificar mediante la secuencia de comandos `request`. Ésta es una de las razones fundamentales para la popularidad de esta aproximación. Sin embargo, los inconvenientes son crónicos y debe considerar esta configuración como último recurso.

Ejemplos: uso de directorios base paramétricos

El archivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
EZDIR=/usr/stuf/EZstuf
HRDDIR=/opt/SUNWstuf/HRDstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert980707141632
```

El archivo pkgmap

```
: 1 1758
1 d none $EZDIR 0775 root bin
1 f none $EZDIR/dirdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/usrdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/filedel 0555 bin bin 40 773 751310229
1 d none $HRDDIR 0775 root bin
1 f none $HRDDIR/mksmart 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mktall 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkcute 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f none /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Administración del directorio base

Cualquier paquete que esté disponible en diversas versiones o para diversas arquitecturas se debe diseñar para *recorrer* el directorio base, si fuera necesario. Recorrer un directorio base significa que si una versión anterior o una arquitectura diferente del paquete que se está instalando ya existe en el directorio base, el paquete que se está instalando resuelve este problema, quizá mediante la creación de un nuevo directorio base con un nombre ligeramente diferente. Las secuencias de comandos `request` y `checkinstall` en Solaris 2.5 y versiones compatibles tienen la posibilidad de modificar la variable de entorno `BASEDIR`. Esto no se aplica a versiones anteriores del sistema operativo Oracle Solaris.

Incluso en versiones anteriores del sistema operativo Oracle Solaris, la secuencia de comandos `request` tenía la autoridad de redefinir directorios en la base de instalación. La secuencia de comandos `request` puede hacerlo de modo que todavía admita las preferencias más administrativas.

Posibilidad de reubicación

Al tiempo que puede seleccionar directorios base para diversos paquetes que se garantizan como exclusivos para una arquitectura y versión, esto conduce a niveles innecesarios de jerarquía de directorios. Por ejemplo, para un producto diseñado para procesadores basados en SPARC y x86, puede organizar los directorios base por procesador y versión tal como se muestra a continuación.

Directorio base	Versión y procesador
`/opt/SUNWstuf/sparc/1.0`	Versión 1.0, SPARC
`/opt/SUNWstuf/sparc/1.2`	Versión 1.2, SPARC
`/opt/SUNWstuf/x86/1.0`	Versión 1.0, x86

Es correcto y funciona, pero considera los nombres y los números como si significaran algo para el administrador. Una mejor aproximación es hacerlo automáticamente *después* de explicarlo al administrador y obtener un permiso.

Esto significa que puede hacer todo el trabajo en el paquete sin que necesite que el administrador lo haga manualmente. Puede asignar el directorio base de forma arbitraria y después establecer de forma transparente los vínculos al cliente adecuado en una secuencia de comandos `postinstall`. También puede usar el comando `pkgadd` para instalar el paquete total o parcialmente en los clientes de la secuencia de comandos `postinstall`. Incluso puede preguntar al administrador qué usuarios o clientes necesitan conocer este paquete y actualizar automáticamente las variables de entorno `PATH` y los archivos `/etc`. Esto es completamente aceptable siempre que, haga lo que haga el paquete tras la instalación, lo deshaga tras la eliminación.

Recorrido de directorios base

Puede beneficiarse de dos métodos para controlar el directorio base en el tiempo de la instalación. Lo primero es mejor para los nuevos paquetes que se instalarán sólo en Solaris 2.5 y versiones compatibles; ofrece información muy útil al administrador, admite varias arquitecturas y versiones instaladas, y requiere una cantidad mínima de trabajo específico. Cualquier paquete puede usar el segundo método y utiliza el control inherente de la secuencia de comandos `request` sobre los parámetros de generación para asegurar instalaciones correctas.

Uso del parámetro BASEDIR

La secuencia de comandos `checkinstall` puede seleccionar el directorio base apropiado en el tiempo de la instalación, lo que significa que el directorio base se puede situar en una posición muy baja del árbol de directorios. Este ejemplo aumenta el directorio base secuencialmente, lo que lleva a directorios con el formato `/opt/SUNWstuf`, `/opt/SUNWstuf.1` y `/opt/SUNWstuf.2`. El administrador puede usar el comando `pkginfo` para determinar qué arquitectura y versión se instalan en cada directorio base.

Si el paquete `SUNWstuf` (que contiene un conjunto de utilidades que hacen el trabajo) usa este método, los archivos `pkginfo` y `pkgmap` tendrían este aspecto.

El archivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt/SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

El archivo pkgmap

```
: 1 1758
1 d none EZstuf 0775 root bin
1 f none EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none HRDstuf 0775 root bin
1 f none HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

Ejemplo: secuencias de comandos de análisis que recorren un BASEDIR

Suponga que la versión x86 de SUNWstuf ya está instalada en el servidor en /opt/SUNWstuf. Cuando el administrador use el comando pkgadd para instalar la versión SPARC, la secuencia de comandos request debe detectar la existencia de la versión x86 e interactuar con el administrador respecto a la instalación.

Nota – El directorio base se puede recorrer sin interacción del administrador en una secuencia de comandos checkinstall, pero si las operaciones arbitrarias como ésta ocurren con demasiada frecuencia, los administradores pierden confianza en el proceso.

Las secuencias de comandos request y checkinstall de un paquete que gestionan esta situación pueden tener este aspecto.

La secuencia de comandos request

```
# request script
for SUNWstuf to walk the BASEDIR parameter.

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

GENMSG="The base directory $LOCAL_BASE already contains a \
different architecture or version of $PKG."

OLDMSG="If the option \"-a none\" was used, press the  \
key and enter an unused base directory when it is requested."

OLDPROMPT="Do you want to overwrite this version? "

OLDHELP="\"y\" will replace the installed package, \"n\" will \
stop the installation."

SUSPEND="Suspending installation at user request using error \
code 1."

MSG="This package could be installed at the unused base directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed base directory? "

HELP="A response of \"y\" will install to the proposed directory and continue,
\"n\" will request a different directory. If the option \"-a none\" was used,
press the  key and enter an unused base directory when it is requested."

DIRPROMPT="Select a preferred base directory ($WRKNG_BASE) "

DIRHELP="The package $PKG will be installed at the location entered."

NUBD_MSG="The base directory has changed. Be sure to update \
any applicable search paths with the actual location of the \
binaries which are at $WRKNG_BASE/EZstuf and $WRKNG_BASE/HRDstuf."

OldSolaris=""
```

```
Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0
      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older version it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR
            OldSolaris="true"
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
fi

WRKNG_BASE=$LOCAL_BASE

    # See if the base directory is already in place and walk it if
    # possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
               # So this is a different architecture or
               # version than what is already there.
               # Walk the base directory
               Suffix=`expr $Suffix + 1`
               WRKNG_BASE=$LOCAL_BASE.$Suffix
               Changed="true"
         fi
done

    # So now we can propose a base directory that isn't claimed by
    # any of our other versions.
if [ $Changed ]; then
         puttext "$GENMSG"
         if [ $OldSolaris ]; then
               puttext "$OLDMSG"
               result=`ckyorn -Q -d "a" -h "$OLDHELP" -p "$OLDPROMPT"`
               if [ $result="n" ]; then
                     puttext "$SUSPEND"
                     exit 1    # suspend installation
               else
                     exit 0
               fi
         else    # The latest functionality is available
               puttext "$MSG"
```

```
                result=‘ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"‘
                if [ $? -eq 3]; then
                      echo quitinstall >> $1
                      exit 0
                fi

                if [ $result="n" ]; then
                      WRKNG_BASE=‘ckpath -ayw -d "$WRKNG_BASE" \
                      -h "$DIRHELP" -p "$DIRPROMPT"‘
                else if [ $result="a" ]
                      exit 0
                fi
           fi
           echo "BASEDIR=$WRKNG_BASE" >> $1
           puttext "$NUBD_MSG"
      fi
fi
exit 0
```

La secuencia de comandos checkinstall

```
# checkinstall
script for SUNWstuf to politely suspend

grep quitinstall $1
if [ $? -eq 0 ]; then
    exit 3        # politely suspend installation
fi

exit 0
```

Esta aproximación no funcionaría muy bien si el directorio base fuera simplemente `/opt`. Este paquete debe indicar el `BASEDIR` de forma más precisa, puesto que `/opt` sería difícil de recorrer. De hecho, según el esquema de montaje, puede que no sea posible. El ejemplo recorre el directorio base mediante la creación de un directorio en `/opt`, lo cual no presenta problemas.

Este ejemplo usa las secuencias de comandos `request` y `checkinstall`, aunque las versiones de Oracle Solaris anteriores a la versión 2.5 no pueden ejecutar una secuencia de comandos `checkinstall`. La secuencia de comandos `checkinstall` de este ejemplo se usa para detener correctamente la instalación, en respuesta a un mensaje privado en forma de cadena `quitinstall`. ". Si esta secuencia de comandos se ejecuta en la versión Solaris 2.3, la secuencia de comandos `checkinstall` se ignora y la secuencia de comandos `request` detiene la instalación con un mensaje de error.

Recuerde que antes de Solaris 2.5 y versiones compatibles, el parámetro `BASEDIR` es de sólo lectura y no se puede cambiar por la secuencia de comandos `request`. Por este motivo, si se detecta una versión antigua del sistema operativo SunOS (mediante la prueba de una variable de entorno `CLIENT_BASEDIR`), la secuencia de comandos `request` sólo tiene dos opciones: continuar o salir.

Uso de rutas paramétricas relativas

Si el producto de software puede instalarse en versiones anteriores del sistema operativo SunOS, la secuencia de comandos `request` debe hacer todo el trabajo necesario. Esta aproximación también se puede usar para manipular varios directorios. Si se necesitan directorios adicionales, se deben incluir en un único directorio base con el fin de ofrecer un producto de fácil administración. Si bien el parámetro `BASEDIR` no ofrece el nivel de granularidad disponible en la última versión de Oracle Solaris, el paquete todavía puede recorrer el directorio base mediante el uso de la secuencia de comandos `request` para manipular las rutas paramétricas. Éste es el aspecto que podrían tener los archivos `pkginfo` y `pkgmap`.

El archivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
SUBBASE=SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

El archivo pkgmap

```
: 1 1758
1 d none $SUBBASE/EZstuf 0775 root bin
1 f none $SUBBASE/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none $SUBBASE/HRDstuf 0775 root bin
1 f none $SUBBASE/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

Este ejemplo no es perfecto. Un comando `pkginfo -r` devuelve `/opt` para la base de instalación, que es bastante ambigua. Muchos paquetes se encuentran en `/opt`, pero al menos se trata de un

directorio con sentido. Al igual que el ejemplo anterior, el siguiente admite completamente diversas arquitecturas y versiones. La secuencia de comandos `request` se puede ajustar a las necesidades del paquete específico y resolver las dependencias aplicables.

Ejemplo: una secuencia de comandos `request` que recorra una ruta paramétrica relativa

```
# request script
for SUNWstuf to walk a parametric path

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

MSG="The target directory $LOCAL_BASE already contains \
different architecture or version of $PKG. This package \
could be installed at the unused target directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed directory? "

HELP="A response of \"y\" will install to the proposed directory \
and continue, \"n\" will request a different directory. If \
the option \"-a none\" was used, press the <RETURN> key and \
enter an unused base directory when it is requested."

DIRPROMPT="Select a relative target directory under $BASEDIR/"

DIRHELP="The package $PKG will be installed at the location entered."

SUSPEND="Suspending installation at user request using error \
code 1."

NUBD_MSG="The location of this package is not the default. Be \
sure to update any applicable search paths with the actual \
location of the binaries which are at $WRKNG_BASE/EZstuf \
and $WRKNG_BASE/HRDstuf."

Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0

      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older versions it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR/$SUBBASE
```

```
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR/$SUBBASE
      fi

WRKNG_BASE=$LOCAL_BASE

# See if the base directory is already in place and walk it if
# possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
               # So this is a different architecture or
               # version than what is already there.
               # Walk the base directory
               Suffix=`expr $Suffix + 1`
               WRKNG_BASE=$LOCAL_BASE.$Suffix
               Changed="true"
         fi
done

# So now we can propose a base directory that isn't claimed by
# any of our other versions.
if [ $Changed ]; then
         puttext "$MSG"
         result=`ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"`
         if [ $? -eq 3 ]; then
               puttext "$SUSPEND"
               exit 1
         fi

         if [ $result="n" ]; then
               WRKNG_BASE=`ckpath -lyw -d "$WRKNG_BASE" -h "$DIRHELP" \
               -p "$DIRPROMPT"`

            elif [ $result="a" ]; then
                  exit 0
            else
                  exit 1
            fi
            echo SUBBASE=$SUBBASE.$Suffix >> $1
            puttext "$NUBD_MSG"
      fi
fi
exit 0
```

Admisión de la reubicación en un entorno heterogéneo

El concepto original tras la creación de paquetes de System V ha supuesto una arquitectura por sistema. El concepto de un servidor no representa una función en el diseño. Ahora, por supuesto, un único servidor puede ofrecer compatibilidad para diversas arquitecturas, lo cual significa que puede haber varias copias del mismo software en un servidor, cada una para una arquitectura diferente. Si bien los paquetes de Oracle Solaris se separan dentro de los límites

recomendados del sistema de archivos (por ejemplo, `/` y `/usr`), con bases de datos de productos en el servidor, así como cada cliente, no todas las instalaciones admiten necesariamente esta división. Determinadas implementaciones admiten una estructura completamente diferente e implican una base de datos de productos habitual. Mientras que dirigir a los clientes a diferentes versiones es algo sencillo, la instalación de paquetes de System V en diversos directorios base puede presentar problemas para el administrador.

Cuando diseña el paquete, también debe tener en cuenta los métodos habituales que los administradores usan para introducir nuevas versiones de software. Los administradores buscan con frecuencia instalar y probar la última versión, de forma paralela a la versión actualmente instalada. El procedimiento implica la instalación de la nueva versión en un directorio base diferente de la versión actual y la orientación de un determinado número de clientes que no sean fundamentales a la nueva versión como prueba. A medida que crece la confianza el administrador redirige cada vez más clientes a la nueva versión. Finalmente, el administrador conserva la versión anterior sólo para emergencias y después, finalmente, la suprime.

Lo que esto significa es que los paquetes destinados a sistemas heterogéneos modernos deben admitir la reubicación total, en el sentido de que el administrador pueda colocarlos en un lugar razonable del sistema de archivos y ver todavía todas sus funciones. Solaris 2.5 y las versiones compatibles ofrecen diversas herramientas que permiten varias arquitecturas y versiones para realizar la instalación sin problemas en el mismo sistema. Solaris 2.4 y las versiones compatibles también admiten la reubicación completa pero la ejecución de la tarea no es tan sencilla.

Aproximación tradicional

Paquetes reubicables

System V ABI implica que la intención original del paquete reubicable era facilitar la instalación del paquete al administrador. En estos momentos, la necesidad de paquetes reubicables va mucho más allá. La comodidad no es la única cuestión; es muy probable que durante la instalación ya haya instalado un producto de software activo en el directorio predeterminado. Un paquete que no esté diseñado para lidiar con esta situación sobrescribe el producto existente o no consigue instalarse. Sin embargo, un paquete diseñado para gestionar diversas arquitecturas y versiones puede instalarse sin problemas y ofrecer al administrador una amplia gama de opciones que sean totalmente compatibles con las tradiciones administrativas existentes.

En algunos aspectos, el problema de varias arquitecturas y el de varias versiones es el mismo. Debe ser posible instalar una variante del paquete existente junto a otras variantes, así como dirigir a los clientes o los consumidores autónomos de sistemas de archivos exportados a una de esas variantes, sin que las funciones se vean comprometidas. Mientras Sun ha establecido métodos para lidiar con varias arquitecturas en un servidor, es posible que el administrador no pueda seguir esas recomendaciones. Todos los paquetes deben ser capaces de cumplir con los deseos razonables de los administradores respecto a la instalación.

Ejemplo: paquete reubicable tradicional

Este ejemplo muestra el aspecto que puede tener un paquete reubicable tradicional. Este paquete se debe ubicar en `/opt/SUNWstuf`, y los archivos `pkginfo` y `pkgmap` pueden tener este aspecto.

El archivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

El archivo pkgmap

```
: 1 1758
1 d none SUNWstuf 0775 root bin
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Este método se conoce como tradicional porque cada objeto del paquete se instala en el directorio base definido por el parámetro `BASEDIR` del archivo `pkginfo`. Por ejemplo, el primer objeto del archivo `pkgmap` se instala como directorio `/opt/SUNWstuf`.

Paquetes absolutos

Un paquete absoluto es el que se instala en un sistema de archivos raíz concreto (`/`). Estos paquetes son difíciles de gestionar desde el punto de vista de varias versiones y arquitecturas. Como norma general, todos los paquetes deben ser reubicables. Sin embargo, hay muy buenos motivos para incluir elementos absolutos en un paquete reubicable.

Ejemplo: paquete absoluto tradicional

Si el paquete `SUNWstuf` fuera absoluto, el parámetro `BASEDIR` no se debería definir en el archivo `pkginfo`, y el archivo `pkgmap` tendría este aspecto.

El archivo pkgmap

```
: 1 1758
1 d none /opt ? ? ?
1 d none /opt/SUNWstuf 0775 root bin
1 d none /opt/SUNWstuf/EZstuf 0775 root bin
1 f none /opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none /opt/SUNWstuf/HRDstuf 0775 root bin
1 f none /opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

En este ejemplo, si el administrador ha especificado un directorio base alternativo durante la instalación, el comando `pkgadd` lo ignoraría. Este paquete siempre se instala en el directorio `/opt/SUNWstuf` del sistema de destino.

El argumento `-R` del comando `pkgadd` funciona según lo esperado. Por ejemplo,

```
pkgadd -d . -R /export/opt/client3 SUNWstuf
```

instala los objetos en `/export/opt/client3/opt/SUNWstuf`, pero esto es lo más cerca que puede estar este paquete de ser reubicable.

Observe el uso del signo de interrogación (`?`) para el directorio `/opt` en el archivo `pkgmap`. Indica que los atributos existentes no se deben cambiar. No significa "crear el directorio con atributos predeterminados" aunque en determinadas circunstancias esto pueda ocurrir. Cualquier directorio que sea específico del nuevo paquete debe especificar todos los atributos de forma explícita.

Paquetes compuestos

Un paquete que contenga objetos reubicables es un paquete reubicable. Esto puede llegar a ser confuso porque un paquete reubicable puede contener rutas absolutas en su archivo `pkgmap`. El uso de una entrada raíz (`/`) en un archivo `pkgmap` puede mejorar los aspectos reubicables del paquete. Los paquetes que tienen entradas raíz y reubicables reciben el nombre de paquetes *compuestos*.

Ejemplo: solución tradicional

Suponga que un objeto del paquete `SUNWstuf` es una secuencia de comandos de inicio que se ejecuta en el nivel de ejecución 2. El archivo `/etc/rc2.d/S70dostuf` se debe instalar como parte del paquete, pero no se puede colocar en el directorio base. Si se supone que un paquete reubicable es la única solución, los archivos `pkginfo` y `pkgmap` podrían tener este aspecto.

El archivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

El archivo pkgmap

```
: 1 1758
1 d none opt/SUNWstuf/EZstuf 0775 root bin
1 f none opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none opt/SUNWstuf/HRDstuf 0775 root bin
1 f none opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none etc    ? ? ?
1 d none etc/rc2.d ? ? ?
1 f none etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

No hay mucha diferencia entre esta aproximación y la de un paquete absoluto. En realidad, resultaría mejor como paquete absoluto: si el administrador proporcionara un directorio base alternativo para este paquete, no funcionaría.

De hecho, sólo un archivo de este paquete debe estar relacionado con la raíz; el resto se puede mover a cualquier lugar. La forma de solucionar este problema mediante el uso de un paquete compuesto se trata a lo largo de esta sección.

Más allá de la tradición

La aproximación que se describe en esta sección no se aplica a todos los paquetes, pero mejora el rendimiento durante la instalación en un entorno heterogéneo. Sólo una pequeña parte se aplica a los paquetes que se han entregado como parte del sistema operativo Oracle Solaris (paquetes integrados). Sin embargo, los paquetes no integrados pueden practicar un empaquetado no tradicional.

El motivo que se esconde tras el fomento de los paquetes reubicables es apoyar a este requisito:

Cuando un paquete se agrega o se suprime, los comportamientos deseables existentes de los productos de software instalados quedarán sin cambios.

Los paquetes no integrados deben residir en `/opt` para asegurar que el nuevo paquete no interfiera con productos existentes.

Otra mirada a los paquetes compuestos

Hay dos normas que seguir a la hora de construir un paquete compuesto funcional:

- Establecer el directorio base según donde va la amplia mayoría de los objetos del paquete.
- Si un objeto del paquete va a un directorio común que no es el directorio base (por ejemplo, `/etc`), especifíquelo como nombre de ruta absoluta en el archivo `prototype`.

En otras palabras: puesto que "reubicable" significa que el objeto se puede instalar en cualquier lugar y que siga funcionando, ninguna secuencia de comandos de inicio ejecutada por `init` en el tiempo del inicio se puede considerar reubicable. Mientras no haya nada incorrecto en la especificación de `/etc/passwd` como ruta relativa en el paquete entregado, sólo hay un lugar al que puede ir.

Cómo conseguir que los nombres de rutas absolutas tengan el aspecto de reubicables

Si va a construir un paquete compuesto, las rutas absolutas deben funcionar de forma que no interfieran en el software instalado existente. Un paquete que se puede incluir por completo en `/opt` evita este problema porque no hay archivos en la ruta. Cuando un archivo de `/etc` se incluye en el paquete, debe asegurarse de que los nombres de la ruta absoluta se comportan del mismo modo que se espera de los nombres de rutas relativa. Observe los dos ejemplos siguientes.

Ejemplo: modificación de un archivo

Descripción

Se agrega una entrada a una tabla, o bien el objeto es una tabla nueva que probablemente modificarán otros programas o paquetes.

Implementación

Defina el objeto como tipo de archivo `e` y perteneciente a la clase `build`, `awk` o `sed`. La secuencia de comandos que ejecuta esta tarea debe suprimirse igual de eficazmente que cuando se agrega.

Ejemplo

Se debe agregar una entrada al archivo `/etc/vfstab` en apoyo del nuevo disco duro de estado sólido.

La entrada del archivo `pkgmap` podría ser

```
1 e sed /etc/vfstab ? ? ?
```

La secuencia de comandos `request` pregunta al operador si el paquete debe modificar el archivo `/etc/vfstab`. Si el operador responde “no”, la secuencia de comandos de solicitud imprimirá las instrucciones para hacer el trabajo manualmente y ejecutará

```
echo "CLASSES=none" >> $1
```

Si el operador responde "sí" se ejecuta

```
echo "CLASSES=none sed" >> $1
```

que activa la secuencia de comandos de acción de clase que realizará las modificaciones necesarias. La clase `sed` significa que el archivo de paquete `/etc/vfstab` es un programa `sed` que contiene las operaciones de instalación y eliminación para el archivo con el mismo nombre en el sistema de destino.

Ejemplo: creación de un nuevo archivo

Descripción

El objeto es un archivo completamente nuevo que no es probable que se modifique posteriormente, o bien sustituye a un archivo propiedad de otro paquete.

Implementación

Defina el objeto del paquete como tipo de archivo `f` e instálelo mediante una secuencia de comandos de acción de clase capaz de deshacer el cambio.

Ejemplo

Se precisa un archivo completamente nuevo en `/etc` para proporcionar la información necesaria con el fin de que se pueda admitir el disco duro de estado sólido, que recibe el nombre de `/etc/shdisk.conf` . La entrada del archivo `pkgmap` podría tener este aspecto:

```
.
.
.
1 f newetc /etc/shdisk.conf
.
.
.
```

La secuencia de comandos de acción de clase `i.newetc` es responsable de la instalación de éste y otros archivos que deben ir en `/etc`. Hace comprobaciones para asegurarse de que no haya otro archivo allí. Si no hay, simplemente copiará el archivo nuevo en su lugar. Si ya hay un archivo en su lugar, hará una copia de seguridad de él antes de instalar el archivo nuevo. La secuencia de comandos `r.newetc` suprime estos archivos y restaura los originales, si fuera necesario. Aquí se encuentra el fragmento clave de la secuencia de comandos de instalación.

```
# i.newetc
while read src dst; do
      if [ -f $dst ]; then
        dstfile=‘basename $dst‘
        cp $dst $PKGSAV/$dstfile
      fi
      cp $src $dst
done

if [ "${1}" = "ENDOFCLASS" ]; then
      cd $PKGSAV
      tar cf SAVE.newetc .
      $INST_DATADIR/$PKG/install/squish SAVE.newetc
fi
```

Observe que esta secuencia de comandos usa la variable de entorno `PKGSAV` para almacenar una copia de seguridad del archivo que se debe sustituir. Cuando el argumento `ENDOFCLASS` pasa a la secuencia de comandos, se trata del comando `pkgadd` que informa a la secuencia de comandos de que éstas son las últimas entradas de esta clase; en ese momento la secuencia de comandos archiva y comprime los archivos que se guardaron mediante un programa de compresión privado almacenado en el directorio de instalación del paquete.

Mientras el uso de la variable de entorno `PKGSAV`, no es fiable durante la actualización de un paquete; si el paquete no se actualiza (mediante un parche, por ejemplo) el archivo de copia de seguridad es seguro. La siguiente secuencia de comandos de eliminación incluye un código para gestionar los demás problemas: el hecho de que versiones más antiguas del comando `pkgrm` no pasen a las secuencias de comandos la ruta correcta a la variable de entorno `PKGSAV`.

La secuencia de comandos de eliminación podría tener este aspecto.

```
# r.newetc

# make sure we have the correct PKGSAV
if [ -d $PKG_INSTALL_ROOT$PKGSAV ]; then
      PKGSAV="$PKG_INSTALL_ROOT$PKGSAV"
fi
```

```
# find the unsquish program
UNSQUISH_CMD=‘dirname $0‘/unsquish

while read file; do
      rm $file
done

if [ "${1}" = ENDOFCLASS ]; then
      if [ -f $PKGSAV/SAVE.newetc.sq ]; then
         $UNSQUISH_CMD $PKGSAV/SAVE.newetc
      fi

      if [ -f $PKGSAV/SAVE.newetc ]; then
         targetdir=dirname $file    # get the right directory
         cd $targetdir
            tar xf $PKGSAV/SAVE.newetc
            rm $PKGSAV/SAVE.newetc
      fi
fi
```

Esta secuencia de comandos usa un algoritmo desinstalado privado (`unsquish`) que se encuentra en el directorio de instalación de la base de datos del paquete. Esto se hace de manera automática mediante el comando `pkgadd` en el tiempo de la instalación. Todas las secuencias de comandos que no estén específicamente reconocidas como de sólo instalación por parte del comando `pkgadd` quedan en este directorio para que lo pueda usar el comando `pkgrm`. No puede determinar dónde se encuentra ese directorio, pero puede confiar en que está completamente disponible y que contiene todas las secuencias de comandos de instalación y los archivos de información adecuados para el paquete. Esta secuencia de comandos busca el directorio por virtud del hecho de que se garantiza la ejecución de la secuencia de comandos de acción de clase a partir del directorio que contiene el programa `unsquish`.

Tenga en cuenta, asimismo, que esta secuencia de comandos no sólo supone que el directorio de destino sea `/etc`. Puede que en realidad sea `/export/root/client2/etc`. El directorio correcto se puede construir de una de estas dos formas.

- Use la construcción `${PKG_INSTALL_ROOT}/etc`, o bien.
- Tome el nombre de directorio de un archivo aprobado por el comando `pkgadd` (lo que esta secuencia de comandos hace).

Mediante el uso de esta aproximación para cada objeto absoluto del paquete, puede estar seguro que el comportamiento deseable actual permanece sin cambios o al menos es recuperable.

Ejemplo: un paquete compuesto

Éste es un ejemplo de los archivos `pkginfo` y `pkgmap` para un paquete compuesto.

El archivo pkginfo

```
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
```

```
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

El archivo pkgmap

```
: 1 1758
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 e daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i i.daemon 509 39560 752978103
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i r.daemon 320 24573 742152591
```

Mientras `S70dostuf` pertenece a la clase `daemon`, los directorios que llevan hasta él (que ya estén en posición en el tiempo de la instalación) pertenecen a la clase `none`. Incluso si los directorios eran exclusivos para este paquete, debe dejarlos en la clase `none`. El motivo es que los directorios se deben crear en primer lugar y después suprimir; siempre ocurre así en el caso de la clase `none`. El comando `pkgadd` crea directorios; no se copian del paquete y no se pasan a una secuencia de comandos de acción de clase que se deba crear. En su lugar, los crea el comando `pkgadd` antes de que llame a la secuencia de comandos de acción de clase de instalación; el comando `pkgrm` suprime directorios después de terminar de suprimir la secuencia de comandos de acción de clase.

Significa que si un directorio de una clase especial contiene objetos de la clase `none`, cuando el comando `pkgrm` intenta suprimir el directorio, se produce un error porque el directorio no estará vacío en ese momento. Si se va a insertar un objeto de clase `none` en un directorio de alguna clase especial, ese directorio no existirá a tiempo de aceptar el objeto. El comando `pkgadd` creará el directorio de forma paralela a la instalación del objeto y puede que no sea capaz de sincronizar los atributos de ese directorio cuando finalmente vea la definición `pkgmap`.

Nota – Al asignar un directorio a una clase, recuerde siempre el orden de creación y eliminación.

Creación de paquetes de instalación remota

Todos los paquetes *deben* ser instalables a distancia. Instalable a distancia significa que no supone que el administrador que instala el paquete esté llevando a cabo la instalación en el sistema de archivos raíz (`/`) del sistema que ejecuta el comando `pkgadd`. Si en una de las secuencias de comandos de procedimientos necesita acceder al archivo `/etc/vfstab` del sistema de destino, debe usar la variable de entorno `PKG_INSTALL_ROOT`. En otras palabras, el nombre de ruta `/etc/vfstab` lo conducirá al archivo `/etc/vfstab` del sistema que ejecuta el comando `pkgadd`, pero es posible que el administrador lleve a cabo la instalación en un cliente en `/export/root/client3`. Se garantiza que la ruta `${PKG_INSTALL_ROOT}/etc/vfstab` lo llevará al sistema de archivos de destino.

Ejemplo: instalación en un sistema cliente

En este ejemplo, el paquete `SUNWstuf` se instala en `client3`, que se configura con `/opt` en su sistema de archivos raíz (`/`). Ya hay otra versión de este paquete instalada en `client3`, y el directorio base está configurado en `basedir=/opt/$PKGINST` desde un archivo de administración, `thisadmin`. (Para obtener más información sobre los archivos de administración, consulte "El archivo administrativo predeterminado" en la página 132). El comando `pkgadd` ejecutado en el servidor es:

```
# pkgadd -a thisadmin -R /export/root/client3 SUNWstuf
```

La tabla siguiente muestra las variables de entorno y sus valores que pasan a las secuencias de comandos de procedimientos.

TABLA 6–1 Valores pasados a secuencias de comandos de procedimientos

Variable de entorno	Valor
`PKGINST`	`SUNWstuf.2`
`PKG_INSTALL_ROOT`	`/export/root/client3`
`CLIENT_BASEDIR`	`/opt/SUNWstuf.2`
`BASEDIR`	`/export/root/client3/opt/SUNWstuf.2`

Ejemplo: instalación en un servidor o sistema autónomo

Para llevar a cabo la instalación en un servidor o sistema autónomo en las mismas circunstancias que el ejemplo anterior, el comando es:

```
# pkgadd -a thisadmin SUNWstuf
```

La tabla siguiente muestra las variables de entorno y sus valores que pasan a las secuencias de comandos de procedimientos.

TABLA 6–2 Valores pasados a secuencias de comandos de procedimientos

Variable de entorno	Valor
PKGINST	SUNWstuf.2
PKG_INSTALL_ROOT	Sin definir.
CLIENT_BASEDIR	/opt/SUNWstuf.2
BASEDIR	/opt/SUNWstuf.2

Ejemplo: montaje de sistemas de archivos compartidos

Suponga que el paquete `SUNWstuf` crea y comparte un sistema de archivos en el servidor en `/export/SUNWstuf/share`. Cuando el paquete se instala en los sistemas cliente, sus archivos `/etc/vfstab` se deben actualizar para montar este sistema de archivos compartido. En esta situación puede usar la variable `CLIENT_BASEDIR`.

La entrada en el cliente debe presentar el punto de montaje en referencia al sistema de archivos del cliente. Esta línea se debe construir correctamente si la instalación parte del servidor o del cliente. Suponga que el nombre del sistema del servidor es `$SERVER`. Puede ir a `$PKG_INSTALL_ROOT/etc/vfstab` y, mediante el comando `sed` o `awk`, construir la línea siguiente para el archivo `/etc/vfstab` del cliente.

```
$SERVER:/export/SUNWstuf/share - $CLIENT_BASEDIR/usr nfs - yes ro
```

Por ejemplo, en el caso del servidor `universe` y el sistema cliente `client9`, la línea del archivo `/etc/vfstab` del sistema cliente tendría este aspecto:

```
universe:/export/SUNWstuf/share - /opt/SUNWstuf.2/usr nfs - yes ro
```

Si se usan estos parámetros correctamente, la entrada siempre monta el sistema de archivos del cliente, independientemente de si se construye desde el servidor o localmente.

Paquetes de parches

Un parche para un paquete es simplemente un paquete suelto diseñado para sobrescribir determinados archivos del original. No hay un motivo real para la inclusión de un paquete suelto excepto guardar espacio en el medio de entrega. También es posible distribuir todo el paquete original con unos cuantos archivos cambiados, o bien proporcionar acceso al paquete modificado en una red. Siempre que esos archivos nuevos sean de hecho diferentes (los demás archivos no se recompilaron), el comando `pkgadd` instala las diferencias. Revise las directrices siguientes relacionadas con los paquetes de parches.

- Si el sistema es lo suficientemente complejo, es aconsejable establecer un sistema de identificación de parches que asegure que dos parches no sustituyan al mismo archivo en un intento de corregir comportamientos aberrantes diferentes. Por ejemplo, a los números base de parches de Sun se les asignan conjuntos de archivos mutuamente excluyentes de los que son responsables.
- Es necesario que sea posible anular un parche.

Es vital que el número de versión del paquete de parches sea el mismo que el del paquete original. Debe hacer un seguimiento del estado del parche del paquete mediante una entrada aparte del archivo `pkginfo` con el formato.

`PATCH=`*patch_number*

Si se cambia la versión del paquete de un parche, cree otra instancia del paquete para que sea extremadamente difícil administrar el producto con el parche. Este método de parches de instancias progresivos tenían ciertas ventajas en las primeras versiones del sistema operativo Oracle Solaris, pero hace que la administración de los sistemas más complicados sea tediosa.

Todos los parámetros de zona del parche deben coincidir con los parámetros de zona del paquete.

Por lo que respecta a los paquetes que componen el sistema operativo Oracle Solaris, debe haber una única copia del paquete en la base de datos de paquetes, aunque puede haber varias instancias de parches. Con el fin de suprimir un objeto de un paquete instalado (mediante el comando `removef`) debe imaginarse qué instancias son propietarias de ese archivo.

Sin embargo, si el paquete (que no forma parte del sistema operativo Oracle Solaris) debe determinar el nivel de parche de un paquete concreto que *forma* parte del sistema operativo Oracle Solaris, se convierte en un problema que debe resolverse aquí. Las secuencias de comandos de instalación pueden ser bastante grandes sin un impacto significativo puesto que no se almacenan en el sistema de archivos de destino. Gracias a las secuencias de comandos de acción de clase y otras secuencias de comandos de procedimientos, puede guardar archivos cambiados mediante la variable de entorno `PKGSAV` (o en otros directorios, más permanentes) con el fin de permitir la anulación de los parches instalados. También puede supervisar el historial de parches configurando las variables de entorno pertinentes mediante las secuencias de comandos `request`. Las secuencias de comandos de las secciones siguientes asumen que

puede haber varios parches cuyo esquema de numeración conlleve algún significado al aplicarse a un único paquete. En este caso, el número de cada parche representa un subconjunto de archivos con funciones relacionadas en el paquete. Dos números de parches diferentes no pueden cambiar el mismo archivo.

Con el fin de convertir un paquete disperso regular en un paquete de parches, las secuencias de comandos descritas en las secciones siguientes pueden doblarse sencillamente en el paquete. Todas ellas son reconocibles como componentes de paquete estándar con la excepción de las dos últimas, `patch_checkinstall` y `patch_postinstall`. Esas dos secuencias de comandos pueden incorporarse al paquete de anulación, si desea incluir la posibilidad de anular el parche. Las secuencias de comandos son bastante sencillas y sus diversas tareas son sencillas.

Nota – Este método de parches se puede usar para sistemas cliente de parches, pero los directorios raíz cliente del servidor deben contar con los permisos correctos para permitir la lectura al usuario `install` o `nobody`.

La secuencia de comandos `checkinstall`

La secuencia de comandos `checkinstall` verifica que el parche es el adecuado para este paquete concreto. Una vez que se haya confirmado, genera la *lista de parches* y la lista de *información de parches*; después las inserta en el archivo de respuesta para su incorporación a la base de datos de paquetes.

Una lista de parches es aquella donde se enumeran los que tienen afectado el paquete actual. Esta lista de parches se registra en el paquete instalado en el archivo `pkginfo` con una línea que podría tener este aspecto:

`PATCHLIST=`*`patch_id patch_id ...`*

Una lista de información de parches es aquella de la que depende el parche actual. Esta lista de parches también se registra en el archivo `pkginfo` con una línea que podría tener este aspecto:

```
PATCH_INFO_103203-01=Installed... Obsoletes:103201-01 Requires: \ Incompatibles: 120134-01
```

Nota – Estas líneas (y su formato) se declaran como interfaz pública. Las compañías que distribuyen parches para los paquetes de Oracle Solaris deben actualizar esta lista de forma pertinente. Cuando se entrega un parche, cada paquete del parche contiene una secuencia de comandos `checkinstall` que lleva a cabo esta tarea. Esa misma secuencia de comandos `checkinstall` también actualiza otros parámetros específicos de los parches. Ésta es la nueva arquitectura de parches que recibe el nombre de Direct Instance Patching.

En este ejemplo, tanto los paquetes originales como sus parches existen en el mismo directorio. Los dos paquetes originales reciben el nombre de `SUNWstuf.v1` y `SUNWstuf.v2`, y sus parches reciben el nombre de `SUNWstuf.p1` y `SUNWstuf.p2`. Esto significa que puede ser muy difícil para

una secuencia de comandos de procedimientos imaginarse de qué directorio vienen estos archivos, puesto que todo lo que sigue en el nombre del paquete al punto (“.”) se elimina en el parámetro PKG, y la variable de entorno PKGINST se refiere a la instancia instalada, no la instancia de origen. Por lo tanto, las secuencias de comandos de procedimientos pueden encontrar el directorio de origen, la secuencia de comandos checkinstall (que siempre se ejecuta desde el directorio de origen) lleva a cabo la consulta y pasa la ubicación como variable SCRIPTS_DIR. Si hubiese habido un solo paquete en el directorio de origen llamado SUNWstuf, las secuencias de comandos de procedimientos podrían haberla encontrado mediante $INSTDIR/$PKG.

```
# checkinstall script to control a patch installation.
# directory format options.
#
#       @(#)checkinstall 1.6 96/09/27 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

INFO_DIR=‘dirname $0‘
INFO_DIR=‘dirname $INFO_DIR‘    # one level up

NOVERS_MSG="PaTcH_MsG 8 Version $VERSION of $PKG is not installed on this system."
ALRDY_MSG="PaTcH_MsG 2 Patch number $Patch_label is already applied."
TEMP_MSG="PaTcH_MsG 23 Patch number $Patch_label cannot be applied until all \
restricted patches are backed out."

# Read the provided environment from what may have been a request script
. $1

# Old systems can’t deal with checkinstall scripts anyway
if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        exit 0
fi

#
# Confirm that the intended version is installed on the system.
#
if [ "${UPDATE}" != "yes" ]; then
        echo "$NOVERS_MSG"
        exit 3
fi

#
# Confirm that this patch hasn’t already been applied and
# that no other mix-ups have occurred involving patch versions and
# the like.
#
Skip=0
active_base=‘echo $Patch_label | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $Patch_label | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+Patchvers_pfx_lnth) } ’‘
```

```
# Is this a restricted patch?
if echo $active_base | egrep -s "Patchstrict_str"; then
        is_restricted="true"
        # All restricted patches are backoutable
        echo "PATCH_NO_UNDO=" >> $1
else
        is_restricted="false"
fi

for patchappl in ${PATCHLIST}; do
        # Is this an ordinary patch applying over a restricted patch?
        if [ $is_restricted = "false" ]; then
                if echo $patchappl | egrep -s "Patchstrict_str"; then
                        echo "$TEMP_MSG"
                        exit 3;
                fi
        fi

        # Is there a newer version of this patch?
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
                        { print substr($0, match($0, "Patchvers_pfx")\
+Patchvers_pfx_lnth) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        echo "PaTcH_MsG 1 Patch number $Patch_label is \
superceded by the already applied $patchappl."
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        # Not newer, it’s the same
                        if [ "$PATCH_UNCONDITIONAL" = "true" ]; then
                                if [ -d $PKGSAV/$Patch_label ]; then
                                        echo "PATCH_NO_UNDO=true" >> $1
                                fi
                        else
                                echo "$ALRDY_MSG"
                                exit 3;
                        fi
                fi
        fi
done

# Construct a list of applied patches in order
echo "PATCHLIST=${PATCHLIST} $Patch_label" >> $1

#
# Construct the complete list of patches this one obsoletes
#
ACTIVE_OBSOLETES=$Obsoletes_label

if [ -n "$Obsoletes_label" ]; then
        # Merge the two lists
        echo $Obsoletes_label | sed ’y/\ /\n/’ | \
        nawk -v PatchObsList="$PATCH_OBSOLETES" ’
        BEGIN {
                printf("PATCH_OBSOLETES=");
```

```
                PatchCount=split(PatchObsList, PatchObsComp, " ");

                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                Inserted=0;
                for(PatchIndex in PatchObs) {
                        if (PatchObs[PatchIndex] == $0) {
                                if (Inserted == 0) {
                                        PatchObsCnt[PatchIndex]=PatchObsCnt\
[PatchIndex]+1;
                                        Inserted=1;
                                } else {
                                        PatchObsCnt[PatchIndex]=0;
                                }
                        }
                }
                if (Inserted == 0) {
                        printf ("%s@1 ", $0);
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] != 0) {
                                printf("%s@%d ", PatchObs[PatchIndex], \
PatchObsCnt[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1
        # Clear the parameter since it has already been used.
        echo "Obsoletes_label=" >> $1

        # Pass it's value on to the preinstall under another name
        echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $1
fi

#
# Construct PATCH_INFO line for this package.
#

tmpRequire=`nawk -F= ' $1 ~ /REQUIR/ { print $2 } ' $INFO_DIR/pkginfo `
tmpIncompat=`nawk -F= ' $1 ~ /INCOMPAT/ { print $2 } ' $INFO_DIR/pkginfo `

if [ -n "$tmpRequire" ] && [ -n "$tmpIncompat" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
          Incompatibles: $tmpIncompat" >> $1
elif [ -n "$tmpRequire" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
```

```
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
Incompatibles: " >> $1
elif [ -n "$tmpIncompat" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: ‘date‘ From: ‘uname -n‘ \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: \
$tmpIncompat" >> $1
else
        echo "PATCH_INFO_$Patch_label=Installed: ‘date‘ From: ‘uname -n‘ \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: " >> $1
fi

#
# Since this script is called from the delivery medium and we may be using
# dot extensions to distinguish the different patch packages, this is the
# only place we can, with certainty, trace that source for our backout
# scripts. (Usually $INST_DATADIR would get us there).
#
echo "SCRIPTS_DIR=‘dirname $0‘" >> $1

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

La secuencia de comandos `preinstall`

La secuencia de comandos `preinstall` inicializa el archivo `prototype`, los archivos de información y las secuencias de comandos de instalación para que se genere el paquete de anulación. Esta secuencia de comandos es muy sencilla y las secuencias de comandos restantes de este ejemplo sólo permiten un paquete de anulación para describir archivos regulares.

Si quisiera restaurar vínculos simbólicos, vínculos físicos, dispositivos y conducciones con nombre en un paquete de anulación, podría modificar la secuencia de comandos `preinstall` para usar el comando `pkgproto` con el fin de comparar el archivo `pkgmap` entregado con los archivos instalados, y después crear una entrada de archivo `prototype` para cada elemento que no sea un archivo que se deba cambiar en el paquete de anulación. El método que debe usar es parecido al de la secuencia de comandos de acción de clase.

Las secuencias de comandos `patch_checkinstall` y `patch_postinstall` se insertan en el árbol de origen del paquete desde la secuencia de comandos `preinstall`. Estas dos secuencias de comandos deshacen lo que el parche hace.

```
# This script initializes the backout data for a patch package
# directory format options.
#
#       @(#)preinstall 1.5 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
```

```
#

PATH=/usr/sadm/bin:$PATH
recovery="no"

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
    if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] || [ -d \
"$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
        recovery="yes"
    fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$BUILD_DIR/files/reloc
ROOT_DIR=$BUILD_DIR/files/root
PROTO_FILE=$BUILD_DIR/prototype
PKGINFO_FILE=$BUILD_DIR/pkginfo
THIS_DIR=‘dirname $0‘

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#

        exit 0
fi

#
# Unless specifically denied, initialize the backout patch data by
# creating the build directory and copying over the original pkginfo
# which pkgadd saved in case it had to be restored.
#
if [ "$PATCH_NO_UNDO" != "true" ] && [ "$recovery" = "no" ]; then
        if [ -d $BUILD_DIR ]; then
                rm -r $BUILD_DIR
        fi

        # If this is a retry of the same patch then recovery is set to
        # yes. Which means there is a build directory already in
        # place with the correct backout data.

        if [ "$recovery" = "no" ]; then
                mkdir $BUILD_DIR
                mkdir -p $RELOC_DIR
                mkdir $ROOT_DIR
        fi
```

```
#
# Here we initialize the backout pkginfo file by first
# copying over the old pkginfo file and themn adding the
# ACTIVE_PATCH parameter so the backout will know what patch
# it's backing out.
#
# NOTE : Within the installation, pkgparam returns the
# original data.
#
pkgparam -v $PKGINST | nawk '
        $1 ~ /PATCHLIST/        { next; }
        $1 ~ /PATCH_OBSOLETES/  { next; }
        $1 ~ /ACTIVE_OBSOLETES/ { next; }
        $1 ~ /Obsoletes_label/  { next; }
        $1 ~ /ACTIVE_PATCH/     { next; }
        $1 ~ /Patch_label/      { next; }
        $1 ~ /UPDATE/   { next; }
        $1 ~ /SCRIPTS_DIR/      { next; }
        $1 ~ /PATCH_NO_UNDO/    { next; }
        $1 ~ /INSTDATE/ { next; }
        $1 ~ /PKGINST/  { next; }
        $1 ~ /OAMBASE/  { next; }
        $1 ~ /PATH/     { next; }
        { print; } ' > $PKGINFO_FILE
echo "ACTIVE_PATCH=$Patch_label" >> $PKGINFO_FILE
echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $PKGINFO_FILE

# And now initialize the backout prototype file with the
# pkginfo file just formulated.
echo "i pkginfo" > $PROTO_FILE

# Copy over the backout scripts including the undo class
# action scripts
for script in $SCRIPTS_DIR/*; do
        srcscript=`basename $script`
        targscript=`echo $srcscript | nawk '
                { script=$0; }
                /u\./ {
                        sub("u.", "i.", script);
                        print script;
                        next;
                }
                /patch_/ {
                        sub("patch_", "", script);
                        print script;
                        next;
                }
                { print "dont_use" } '`
        if [ "$targscript" = "dont_use" ]; then
                continue
        fi

        echo "i $targscript=$FILE_DIR/$targscript" >> $PROTO_FILE
        cp $SCRIPTS_DIR/$srcscript $FILE_DIR/$targscript
done
 #
# Now add entries to the prototype file that won't be passed to
# class action scripts. If the entry is brand new, add it to the
# deletes file for the backout package.
```

```
        #
        Our_Pkgmap=‘dirname $SCRIPTS_DIR‘/pkgmap
        BO_Deletes=$FILE_DIR/deletes

        nawk -v basedir=${BASEDIR:-/} ’
                BEGIN { count=0; }
                {
                        token = $2;
                        ftype = $1;
                }
                $1 ~ /[#\!:]/ { next; }
                $1 ~ /[0123456789]/ {
                        if ( NF >= 3) {
                                token = $3;
                                ftype = $2;
                        } else {
                                next;
                        }
                }
                { if (ftype == "i" || ftype == "e" || ftype == "f" || ftype == \
"v" || ftype == "d") { next; } }
                {
                        equals=match($4, "=")-1;
                        if ( equals == -1 ) { print $3, $4; }
                        else { print $3, substr($4, 0, equals); }
                }
                ’ < $Our_Pkgmap | while read class path; do
                        #
                        # NOTE: If pkgproto is passed a file that is
                        # actually a hard link to another file, it
                        # will return ftype "f" because the first link
                        # in the list (consisting of only one file) is
                        # viewed by pkgproto as the source and always
                        # gets ftype "f".
                        #
                        # If this isn’t replacing something, then it
                        # just goes to the deletes list.
                        #
                        if valpath -l $path; then
                                Chk_Path="$BASEDIR/$path"
                                Build_Path="$RELOC_DIR/$path"
                                Proto_From="$BASEDIR"
                        else    # It’s an absolute path
                                Chk_Path="$PKG_INSTALL_ROOT$path"
                                Build_Path="$ROOT_DIR$path"
                                Proto_From="$PKG_INSTALL_ROOT"
                        fi
                         #
                        # Hard links have to be restored as regular files.
                        # Unlike the others in this group, an actual
                        # object will be required for the pkgmk.
                        #
                        if [ -f "$Chk_Path" ]; then
                                mkdir -p ‘dirname $Build_Path‘
                                cp $Chk_Path $Build_Path
                                cd $Proto_From
                                pkgproto -c $class "$Build_Path=$path" 1>> \
$PROTO_FILE 2> /dev/null
                                cd $THIS_DIR
```

```
                        elif [ -h "$Chk_Path" -o \
                             -c "$Chk_Path" -o \
                             -b "$Chk_Path" -o \
                             -p "$Chk_Path" ]; then
                                pkgproto -c $class "$Chk_Path=$path" 1>> \
$PROTO_FILE 2> /dev/null
                        else
                                echo $path >> $BO_Deletes
                        fi
                done
fi

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

Secuencia de comandos de acción de clase

La secuencia de comandos de acción de clase crea una copia de cada archivo que sustituye a un archivo existente y agrega una línea correspondiente al archivo `prototype` para el paquete de anulación. Todo esto se lleva a cabo con secuencias de comandos `nawk` bastante sencillas. La secuencia de comandos de acción de clase recibe una lista de pares de origen/destino que se compone de archivos ordinarios que no coinciden con los archivos instalados correspondientes. Los vínculos simbólicos y otros elementos que no sean archivos deben gestionarse en la secuencia de comandos `preinstall`.

```
# This class action script copies the files being replaced
# into a package being constructed in $BUILD_DIR. This class
# action script is only appropriate for regular files that
# are installed by simply copying them into place.
#
# For special package objects such as editable files, the patch
# producer must supply appropriate class action scripts.
#
# directory format options.
#
#       @(#)i.script 1.6 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

ECHO="/usr/bin/echo"
SED="/usr/bin/sed"
PKGPROTO="/usr/bin/pkgproto"
EXPR="/usr/bin/expr"    # used by dirname
MKDIR="/usr/bin/mkdir"
CP="/usr/bin/cp"
RM="/usr/bin/rm"
```

```
MV="/usr/bin/mv"

recovery="no"
Pn=$$
procIdCtr=0

CMDS_USED="$ECHO $SED $PKGPROTO $EXPR $MKDIR $CP $RM $MV"
LIBS_USED=""

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
        if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] ||
\
[ -d "$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
                recovery="yes"
        fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
PROGNAME=‘basename $0‘

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        PATCH_NO_UNDO="true"
fi

# Since this is generic, figure out the class.
Class=‘echo $PROGNAME | nawk ’ { print substr($0, 3)  }’‘

# Since this is an update, $BASEDIR is guaranteed to be correct
BD=${BASEDIR:-/}

cd $BD

#
# First, figure out the dynamic libraries that can trip us up.
#
if [ -z "$PKG_INSTALL_ROOT" ]; then
        if [ -x /usr/bin/ldd ]; then
                LIB_LIST=‘/usr/bin/ldd $CMDS_USED | sort -u | nawk ’
                        $1 ~ /\// { continue; }
                        { printf "%s ", $3 } ’‘
        else
                LIB_LIST="/usr/lib/libc.so.1 /usr/lib/libdl.so.1
\
/usr/lib/libw.so.1 /usr/lib/libintl.so.1 /usr/lib/libadm.so.1 \
/usr/lib/libelf.so.1"
```

```
        fi
fi

#
# Now read the list of files in this class to be replaced. If the file
# is already in place, then this is a change and we need to copy it
# over to the build directory if undo is allowed. If it's a new entry
# (No $dst), then it goes in the deletes file for the backout package.
#
procIdCtr=0
while read src dst; do
        if [ -z "$PKG_INSTALL_ROOT" ]; then
                Chk_Path=$dst
                for library in $LIB_LIST; do
                        if [ $Chk_Path = $library ]; then
                                $CP $dst $dst.$Pn
                                LIBS_USED="$LIBS_USED $dst.$Pn"
                                LD_PRELOAD="$LIBS_USED"
                                export LD_PRELOAD
                        fi
                done
        fi

        if [ "$PATCH_PROGRESSIVE" = "true" ]; then
                # If this is being used in an old-style patch, insert
                # the old-style script commands here.

                #XXXOld_CommandsXXX#
                echo >/dev/null # dummy
        fi

        if [ "${PATCH_NO_UNDO}" != "true" ]; then
                #
                # Here we construct the path to the appropriate source
                # tree for the build. First we try to strip BASEDIR. If
                # there's no BASEDIR in the path, we presume that it is
                # absolute and construct the target as an absolute path
                # by stripping PKG_INSTALL_ROOT. FS_Path is the path to
                # the file on the file system (for deletion purposes).
                # Build_Path is the path to the object in the build
                # environment.
                #
                if [ "$BD" = "/" ]; then
                        FS_Path=`$ECHO $dst | $SED s@"$BD"@@`
                else
                        FS_Path=`$ECHO $dst | $SED s@"$BD/"@@`
                fi

                # If it's an absolute path the attempt to strip the
                # BASEDIR will have failed.
                if [ $dst = $FS_Path ]; then
                        if [ -z "$PKG_INSTALL_ROOT" ]; then
                                FS_Path=$dst
                                Build_Path="$ROOT_DIR$dst"
                        else
                                Build_Path="$ROOT_DIR`echo $dst | \
                                    sed s@"$PKG_INSTALL_ROOT"@@`"
                                FS_Path=`echo $dst | \
                                    sed s@"$PKG_INSTALL_ROOT"@@`
```

```
                fi
        else
                Build_Path="$RELOC_DIR/$FS_Path"
        fi

        if [ -f $dst ]; then    # If this is replacing something
                cd $FILE_DIR
                #
                # Construct the prototype file entry. We replace
                # the pointer to the filesystem object with the
                # build directory object.
                #
                $PKGPROTO -c $Class $dst=$FS_Path | \
                    $SED -e s@=$dst@=$Build_Path@ >> \
                    $BUILD_DIR/prototype

                # Now copy over the file
                if [ "$recovery" = "no" ]; then
                        DirName=‘dirname $Build_Path‘
                        $MKDIR -p $DirName
                        $CP -p $dst $Build_Path
                else
                        # If this file is already in the build area skip it
                        if [ -f "$Build_Path" ]; then
                                cd $BD
                                continue
                        else
                                DirName=‘dirname $Build_Path‘
                                if [ ! -d "$DirName" ]; then
                                        $MKDIR -p $DirName
                                fi
                                $CP -p $dst $Build_Path
                        fi
                fi

                cd $BD
        else    # It’s brand new
                $ECHO $FS_Path >> $BO_Deletes
        fi
fi

# If special processing is required for each src/dst pair,
# add that here.
#
#XXXSpecial_CommandsXXX#
#

$CP $src $dst.$$$procIdCtr
if [ $? -ne 0 ]; then
        $RM $dst.$$$procIdCtr 1>/dev/null 2>&1
else
        $MV -f $dst.$$$procIdCtr $dst
        for library in $LIB_LIST; do
                if [ "$library" = "$dst" ]; then
                        LD_PRELOAD="$dst"
                        export LD_PRELOAD
                fi
        done
fi
```

```
        procIdCtr=‘expr $procIdCtr + 1‘
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

#
# Release the dynamic libraries
#
for library in $LIBS_USED; do
        $RM -f $library
done

exit 0
```

La secuencia de comandos `postinstall`

La secuencia de comandos `postinstall` crea el paquete de anulación mediante la información proporcionada por las demás secuencias de comandos. Puesto que los comandos `pkgmk` y `pkgtrans` no precisan la base de datos de paquetes, se pueden ejecutar en la instalación de un paquete.

En este ejemplo se permite deshacer el parche mediante la construcción de un paquete de formato de flujo en el directorio para guardar (mediante la variable de entorno `PKGSAV)`. No es obvio, pero este paquete debe tener formato de flujo, porque el directorio para guardar cambia de posición durante una operación de `pkgadd`. Si se aplica el comando `pkgadd` a un paquete en su propio directorio para guardar, las suposiciones sobre dónde está el origen del paquete en un momento concreto se vuelven muy poco fiables. Un paquete con formato de flujo se desempaqueta en un directorio temporal y se instala desde allí. (Un paquete con formato de directorio comenzaría la instalación desde el directorio para guardar y se encontraría repentinamente reubicado durante una operación a prueba de fallos de `pkgadd`).

Para determinar qué parches se aplican a un paquete, use este comando:

```
$ pkgparam SUNWstuf PATCHLIST
```

Con la excepción de `PATCHLIST`, una interfaz pública de Sun, no hay nada significativo en los nombres de parámetros en este ejemplo. En lugar de `PATCH`, podría usar la `SUNW_PATCHID` tradicional y se podría cambiar en consonancia el nombre de otras listas como `PATCH_EXCL` y `PATCH_REQD`.

Si determinados paquetes de parches dependen de otros que están disponibles en el mismo medio, la secuencia de comandos `checkinstall` podría determinar esto y crear una secuencia de comandos que se ejecutara mediante la secuencia de comandos `postinstall` del mismo modo que el ejemplo de actualización (consulte “Actualización de paquetes” en la página 175).

```
# This script creates the backout package for a patch package
#
# directory format options.
#
# @(#) postinstall 1.6 96/01/29 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

# Description:
#       Set the TYPE parameter for the remote file
#
# Parameters:
#       none
#
# Globals set:
#       TYPE

set_TYPE_parameter () {
        if [ ${PATCH_UNDO_ARCHIVE:?????} = "/dev" ]; then
                # handle device specific stuff
                TYPE="removable"
        else
                TYPE="filesystem"
        fi
}

#
# Description:
#       Build the remote file that points to the backout data
#
# Parameters:
#       $1:     the un/compressed undo archive
#
# Globals set:
#       UNDO, STATE

build_remote_file () {
        remote_path=$PKGSAV/$Patch_label/remote
        set_TYPE_parameter
        STATE="active"

        if [ $1 = "undo" ]; then
                UNDO="undo"
        else
                UNDO="undo.Z"
        fi

        cat > $remote_path << EOF
# Backout data stored remotely
TYPE=$TYPE
FIND_AT=$ARCHIVE_DIR/$UNDO
STATE=$STATE
EOF
}

PATH=/usr/sadm/bin:$PATH
```

```
if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

if [ ! -n "$PATCH_UNDO_ARCHIVE" ]; then
        PATCH_UNDO_ARCHIVE="none"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
THIS_DIR=‘dirname $0‘
PROTO_FILE=$BUILD_DIR/prototype
TEMP_REMOTE=$PKGSAV/$Patch_label/temp

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # remove the scripts that are left behind
        install_scripts=‘dirname $0‘
        rm $install_scripts/checkinstall \
$install_scripts/patch_checkinstall $install_scripts/patch_postinstall

        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#

        exit 0
fi
#
# At this point we either have a deletes file or we don’t. If we do,
# we create a prototype entry.
#
if [ -f $BO_Deletes ]; then
        echo "i deletes=$BO_Deletes" >> $BUILD_DIR/prototype
fi

#
# Now delete everything in the deletes list after transferring
# the file to the backout package and the entry to the prototype
# file. Remember that the pkgmap will get the CLIENT_BASEDIR path
# but we have to actually get at it using the BASEDIR path. Also
# remember that removef will import our PKG_INSTALL_ROOT
#
Our_Deletes=$THIS_DIR/deletes
if [ -f $Our_Deletes ]; then
        cd $BASEDIR

        cat $Our_Deletes | while read path; do
                Reg_File=0

                if valpath -l $path; then
                        Client_Path="$CLIENT_BASEDIR/$path"
```

```
                        Build_Path="$RELOC_DIR/$path"
                        Proto_Path=$BASEDIR/$path
                else    # It's an absolute path
                        Client_Path=$path
                        Build_Path="$ROOT_DIR$path"
                        Proto_Path=$PKG_INSTALL_ROOT$path
                fi

                # Note: If the file isn't really there, pkgproto
                # doesn't write anything.
                LINE=`pkgproto $Proto_Path=$path`
                ftype=`echo $LINE | nawk '{ print $1 }'`
                if [ $ftype = "f" ]; then
                        Reg_File=1
                fi

                if [ $Reg_File = 1 ]; then
                        # Add source file to the prototype entry
                        if [ "$Proto_Path" = "$path" ]; then
                                LINE=`echo $LINE | sed -e s@$Proto_Path@$Build_Path@2`
                        else
                                LINE=`echo $LINE | sed -e s@$Proto_Path@$Build_Path@`
                        fi

                        DirName=`dirname $Build_Path`
                        # make room in the build tree
                        mkdir -p $DirName
                        cp -p $Proto_Path $Build_Path
                fi

                # Insert it into the prototype file
                echo $LINE 1>>$PROTO_FILE 2>/dev/null

                # Remove the file only if it's OK'd by removef
                rm `removef $PKGINST $Client_Path` 1>/dev/null 2>&1
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Unless specifically denied, make the backout package.
#
if [ "$PATCH_NO_UNDO" != "true" ]; then
        cd $BUILD_DIR   # We have to build from here.

        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                STAGE_DIR="$PATCH_UNDO_ARCHIVE"
                ARCHIVE_DIR="$PATCH_UNDO_ARCHIVE/$Patch_label/$PKGINST"
                mkdir -p $ARCHIVE_DIR
                mkdir -p $PKGSAV/$Patch_label
        else
                if [ -d $PKGSAV/$Patch_label ]; then
                        rm -r $PKGSAV/$Patch_label
                fi
                STAGE_DIR=$PKGSAV
                ARCHIVE_DIR=$PKGSAV/$Patch_label
                mkdir $ARCHIVE_DIR
```

```
        fi

        pkgmk -o -d $STAGE_DIR 1>/dev/null 2>&1
        pkgtrans -s $STAGE_DIR $ARCHIVE_DIR/undo $PKG 1>/dev/null 2>&1
        compress $ARCHIVE_DIR/undo
        retcode=$?
        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                if [ $retcode != 0 ]; then
                        build_remote_file "undo"
                else
                        build_remote_file "undo.Z"
                fi
        fi
        rm -r $STAGE_DIR/$PKG

        cd ..
        rm -r $BUILD_DIR
        # remove the scripts that are left behind
        install_scripts=‘dirname $0‘
        rm $install_scripts/checkinstall $install_scripts/patch_\
checkinstall $install_scripts/patch_postinstall
fi

#
# Since this apparently worked, we’ll mark as obsoleted the prior
# versions of this patch - installpatch deals with explicit obsoletions.
#
cd ${PKG_INSTALL_ROOT:-/}
cd var/sadm/pkg

active_base=‘echo $Patch_label | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘

List=‘ls -d $PKGINST/save/${active_base}*‘
if [ $? -ne 0 ]; then
        List=""
fi

for savedir in $List; do
        patch=‘basename $savedir‘
        if [ $patch = $Patch_label ]; then
                break
        fi

        # If we get here then the previous patch gets deleted
        if [ -f $savedir/undo ]; then
                mv $savedir/undo $savedir/obsolete
                echo $Patch_label >> $savedir/obsoleted_by
        elif [ -f $savedir/undo.Z ]; then
                mv $savedir/undo.Z $savedir/obsolete.Z
                echo $Patch_label >> $savedir/obsoleted_by
        elif  [ -f $savedir/remote ]; then
                ‘grep . $PKGSAV/$patch/remote | sed ’s/STATE=.*/STATE=obsolete/
’ > $TEMP_REMOTE‘
                rm -f $PKGSAV/$patch/remote
                mv $TEMP_REMOTE $PKGSAV/$patch/remote
                rm -f $TEMP_REMOTE
                echo $Patch_label >> $savedir/obsoleted_by
        elif  [ -f $savedir/obsolete -o -f $savedir/obsolete.Z ]; then
```

```
                echo $Patch_label >> $savedir/obsoleted_by
        fi
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

La secuencia de comandos patch_checkinstall

```
# checkinstall script to validate backing out a patch.
# directory format option.
#
#       @(#)patch_checkinstall 1.2 95/10/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

LATER_MSG="PaTcH_MsG 6 ERROR: A later version of this patch is applied."
NOPATCH_MSG="PaTcH_MsG 2 ERROR: Patch number $ACTIVE_PATCH is not installed"
NEW_LIST=""

# Get OLDLIST
. $1

#
# Confirm that the patch that got us here is the latest one installed on
# the system and remove it from PATCHLIST.
#
Is_Inst=0
Skip=0
active_base=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
for patchappl in ${OLDLIST}; do
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
                        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        puttext "$LATER_MSG"
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        Is_Inst=1
                        Skip=1
                fi
        fi
```

```
        if [ $Skip = 1 ]; then
                Skip=0
        else
                NEW_LIST="${NEW_LIST} $patchappl"
        fi
done

if [ $Is_Inst = 0 ]; then
        puttext "$NOPATCH_MSG"
        exit 3
fi

#
# OK, all's well. Now condition the key variables.
#
echo "PATCHLIST=${NEW_LIST}" >> $1
echo "Patch_label=" >> $1
echo "PATCH_INFO_$ACTIVE_PATCH=backed out" >> $1

# Get the current PATCH_OBSOLETES and condition it
Old_Obsoletes=$PATCH_OBSOLETES

echo $ACTIVE_OBSOLETES | sed 'y/\ /\n/' | \
nawk -v PatchObsList="$Old_Obsoletes" '
        BEGIN {
                printf("PATCH_OBSOLETES=");
                PatchCount=split(PatchObsList, PatchObsComp, " ");

                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                for(PatchIndex in PatchObs) {
                        if (PatchObs[PatchIndex] == $0) {
                                PatchObsCnt[PatchIndex]=PatchObsCnt[PatchIndex]-1;
                        }
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] > 0 ) {
                                printf("%s@%d ", PatchObs[PatchIndex], PatchObsCnt\
[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1

        # remove the used parameters
        echo "ACTIVE_OBSOLETES=" >> $1
        echo "Obsoletes_label=" >> $1

exit 0
```

La secuencia de comandos patch_postinstall

```
# This script deletes the used backout data for a patch package
# and removes the deletes file entries.
#
# directory format options.
#
#       @(#)patch_postinstall 1.2 96/01/29 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#
PATH=/usr/sadm/bin:$PATH
THIS_DIR=‘dirname $0‘

Our_Deletes=$THIS_DIR/deletes

#
# Delete the used backout data
#
if [ -f $Our_Deletes ]; then
        cat $Our_Deletes | while read path; do
                if valpath -l $path; then
                        Client_Path=‘echo "$CLIENT_BASEDIR/$path" | sed s@//@/@‘
                else    # It’s an absolute path
                        Client_Path=$path
                fi
                rm ‘removef $PKGINST $Client_Path‘
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Remove the deletes file, checkinstall and the postinstall
#
rm -r $PKGSAV/$ACTIVE_PATCH
rm -f $THIS_DIR/checkinstall $THIS_DIR/postinstall

exit 0
```

Actualización de paquetes

El proceso para actualizar un paquete es muy diferente del proceso para sobrescribirlo. Si bien hay herramientas especiales para admitir la actualización de paquetes estándar distribuidos como parte del sistema operativo Oracle Solaris, se puede diseñar un paquete no integrado para que admita su propia actualización. En diversos ejemplos anteriores, se describieron paquetes que van más allá y controlan el método exacto de instalación bajo la dirección del administrador. Puede diseñar la secuencia de comandos `request` para que admita también la actualización directa de un paquete. Si el administrador elige la instalación de un paquete para que sustituya a otro completamente, sin que queden archivos obsoletos residuales, las secuencias de comandos de paquetes permiten esta posibilidad.

Las secuencias de comandos `request` y `postinstall` de este ejemplo ofrecen un único paquete actualizable. La secuencia de comandos `request` se comunica con el administrador y, a continuación, configura un único archivo en el directorio `/tmp` para suprimir la instancia de paquete anterior. (Aunque la secuencia de comandos `request` cree un archivo (lo cual está prohibido), no hay ningún problema, ya que todos tienen acceso a `/tmp`).

La secuencia de comandos `postinstall` ejecuta a continuación la secuencia de comandos de shell en `/tmp` que ejecuta el comando `pkgrm` necesario relacionado con el paquete anterior y después se suprime.

Este ejemplo ilustra una actualización básica. Son menos de 50 líneas de código, incluidos algunos mensajes bastante largos. Se podría ampliar para anular la actualización o llevar a cabo otras transformaciones principales en el paquete, según lo requiera el diseñador.

El diseño de la interfaz de usuario para una opción de actualización debe estar completamente seguro de que el administrador tiene plena conciencia del proceso y que ha solicitado activamente la actualización en lugar de una instalación paralela. No hay problemas en llevar a cabo una operación compleja bien comprendida como una actualización siempre que la interfaz del usuario deje la operación clara.

La secuencia de comandos `request`

```
# request script
control an upgrade installation

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

UPGRADE_MSG="Do you want to upgrade the installed version ?"

UPGRADE_HLP="If upgrade is desired, the existing version of the \
    package will be replaced by this version. If it is not \
    desired, this new version will be installed into a different \
    base directory and both versions will be usable."

UPGRADE_NOTICE="Conflict approval questions may be displayed. The \
    listed files are the ones that will be upgraded. Please \
    answer \"y\" to these questions if they are presented."

pkginfo -v 1.0 -q SUNWstuf.\*

if [ $? -eq 0 ]; then
      # See if upgrade is desired here
      response=‘ckyorn -p "$UPGRADE_MSG" -h "$UPGRADE_HLP"‘
      if [ $response = "y" ]; then
            OldPkg=‘pkginfo -v 1.0 -x SUNWstuf.\* | nawk ’ \
            /SUNW/{print $1} ’‘
            # Initiate upgrade
            echo "PATH=/usr/sadm/bin:$PATH" > $UPGR_SCRIPT
            echo "sleep 3" >> $UPGR_SCRIPT
            echo "echo Now removing old instance of $PKG" >> \
```

```
            $UPGR_SCRIPT
            if [ ${PKG_INSTALL_ROOT} ]; then
                  echo "pkgrm -n -R $PKG_INSTALL_ROOT $OldPkg" >> \
                  $UPGR_SCRIPT
            else
                  echo "pkgrm -n $OldPkg" >> $UPGR_SCRIPT
            fi
            echo "rm $UPGR_SCRIPT" >> $UPGR_SCRIPT
            echo "exit $?" >> $UPGR_SCRIPT

            # Get the original package's base directory
            OldBD=‘pkgparam $OldPkg BASEDIR‘
            echo "BASEDIR=$OldBD" > $1
            puttext -l 5 "$UPGRADE_NOTICE"
      else
             if [ -f $UPGR_SCRIPT ]; then
                   rm -r $UPGR_SCRIPT
             fi
      fi
fi

exit 0
```

La secuencia de comandos `postinstall`

```
# postinstall
to execute a simple upgrade

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

if [ -f $UPGR_SCRIPT ]; then
      sh $UPGR_SCRIPT &
fi

exit 0
```

Creación de paquetes de archivo de clase

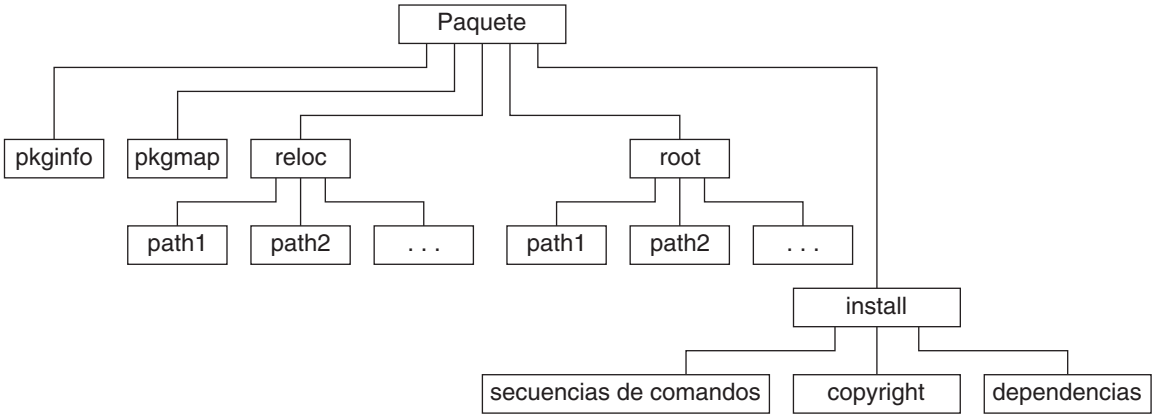
Un paquete de archivos de clase, una mejora en la Application Binary Interface (ABI, Interfaz binaria de aplicaciones), es aquel en que determinados conjuntos de archivos se han combinado en archivos de almacenamiento o archivos sencillos, y que de forma optativa se han comprimido o cifrado. Los formatos de archivos de clase aumentan la velocidad de instalación inicial en un 30% y mejora la fiabilidad durante la instalación de paquetes y parches en sistemas de archivos potencialmente activos.

Las secciones siguientes ofrecen información sobre la estructura de directorios de paquetes de archivos, palabras clave y la utilidad `faspac`.

Estructura del directorio de paquetes de archivos

La entrada de paquete que aparece en la figura siguiente representa el directorio que contiene los archivos de paquetes. Este directorio debe tener el mismo nombre que el paquete.

FIGURA 6–1 Estructura del directorio de paquetes



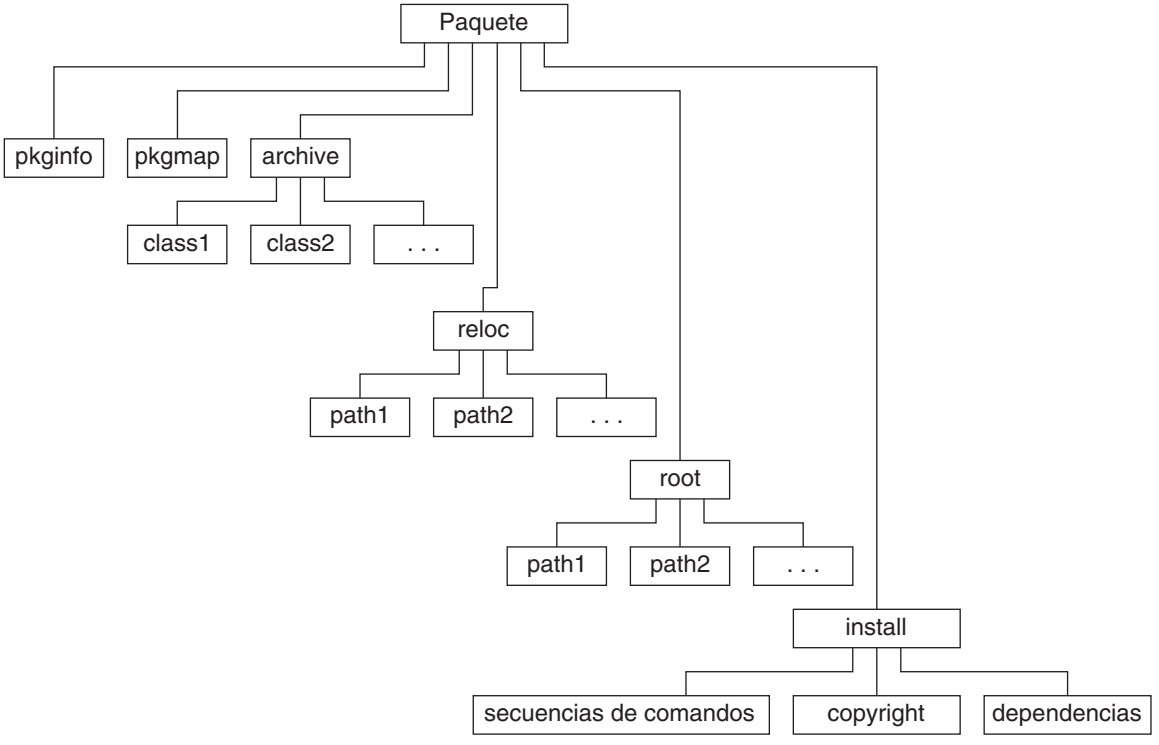
A continuación se enumeran las funciones de los archivos y directorios que se incluyen en el directorio de paquetes.

Elemento	Descripción
`pkginfo`	Archivo que describe el paquete en conjunto que incluye variables de entorno especiales y directivas de instalación
`pkgmap`	Descripción de archivo de cada objeto que instalar, como un archivo, directorio o conducción
`reloc`	Directorio optativo que contiene los archivos que se van a instalar relacionados con el directorio base (objetos reubicables)
`root`	Directorio optativo que contiene los archivos que se van a instalar relacionados con el directorio `root` (objetos raíz)
`install`	Directorio optativo que contiene secuencias de comandos y otros archivos auxiliares (excepto en el caso de `pkginfo` y `pkgmap`, todos los archivos `ftype i` hasta aquí)

El formato de archivo de clase permite al creador de paquetes combinar archivos de los directorios `reloc` y `root` en archivos que se pueden comprimir, cifrar o procesar del modo que se desee para aumentar la velocidad de la instalación, reducir el tamaño del paquete o aumentar la seguridad del paquete.

La ABI permite asignar cualquier archivo de un paquete a una clase. Todos los archivos de una clase específica se pueden instalar en el disco mediante un método personalizado definido por una secuencia de comandos de acción de clase. Este método personalizado puede usar programas disponibles en el sistema de destino o programas distribuidos con el paquete. El formato resultante se parece mucho al formato estándar de la ABI. Tal como se muestra en la ilustración siguiente, se agrega otro directorio. Cualquier clase de archivo diseñado como archivo de almacenamiento se combina en un único archivo y se coloca en el directorio `archive`. Todos los archivos que se hayan archivado se suprimen de los directorios `reloc` y `root` y se coloca una secuencia de comandos de acción de clase de instalación en el directorio `install`.

FIGURA 6–2 Estructura de directorios de paquetes de archivos



Palabras clave para admitir paquetes de archivo de clase

Con el fin de admitir este nuevo formato de archivo de clase, tres nuevas interfaces con formato de palabras clave tienen un significado especial en el archivo `pkginfo`. Estas palabras clave se usan para designar clases que precisen un tratamiento especial. El formato de cada instrucción de palabra clave: `keyword=class1[class2 class3 ...]`. Los valores de cada palabra clave se definen en la tabla siguiente.

Palabra clave	Descripción
`PKG_SRC_NOVERIFY`	Indica a `pkgadd` que no verifique la existencia ni las propiedades de los archivos en los directorios `reloc` o `root` del paquete distribuido si pertenecen a la clase nombrada. Esto es necesario en todas las clases archivadas, porque esos archivos ya no se encuentran en un directorio `reloc` o `root`. Son un archivo de formato privado en el directorio `archive`.
`PKG_DST_QKVERIFY`	Los archivos de estas clases se verifican después de la instalación mediante un algoritmo rápido con poca o ninguna salida de texto. La verificación rápida configura primero los atributos de cada archivo correctamente y después comprueba si la operación ha sido satisfactoria. Después hay una prueba de la hora de modificación y el tamaño del archivo con el `pkgmap`. No se lleva a cabo una verificación `checksum` y la recuperación de errores es más deficiente que la ofrecida por el mecanismo estándar de verificación. En el caso de interrupción de la alimentación o fallo en el disco durante la instalación, es posible que falte coherencia al archivo de contenido respecto a los archivos instalados. Esta falta de coherencia se puede solucionar siempre con un comando `pkgrm`.
`PKG_CAS_PASSRELATIVE`	Por lo general, la secuencia de comandos de acción de clase de instalación recibe de `stdin` una lista de pares de origen y de destino para indicar qué archivos se deben instalar. Las clases asignadas a `PKG_CAS_PASSRELATIVE` no obtienen los pares de origen y de destino. En su lugar reciben una lista única, cuya primera entrada es la ubicación del paquete de origen y el resto son las rutas de destino. La finalidad específica es simplificar la extracción de un archivo. A partir de la ubicación del paquete de origen, puede buscar el archivo en el directorio `archive`. Las rutas de destino pasan a la función responsable de extraer el contenido de un archivo. Cada ruta de destino proporcionada es absoluta o relativa en relación al directorio base, según si la ruta se encontraba originalmente en `root` o `reloc`. Si se elige esta opción, puede que sea difícil combinar las rutas relativas y absolutas en una única clase.

Se precisa una secuencia de comandos de acción de clase para cada clase archivada. Se trata de un archivo que contiene comandos de shell Bourne, ejecutado por `pkgadd` para instalar los archivos desde el archivo. Si se encuentra una secuencia de comandos de acción de clase en el directorio `install` del paquete, `pkgadd` entrega toda la responsabilidad de la instalación a esa secuencia de comandos. La secuencia de comandos de acción de clase se ejecuta con permisos de usuario root y puede situar sus archivos en cualquier lugar del sistema de destino.

Nota – La única palabra clave que es absolutamente necesaria para implementar un paquete de archivo de clase es `PKG_SRC_NOVERIFY`. Las demás se pueden usar para aumentar la velocidad de la instalación o conservar el código.

La utilidad `faspac`

La utilidad `faspac` convierte un paquete ABI estándar en un formato de archivo de clase utilizado para paquetes integrados. Esta utilidad realiza tareas de archivado mediante cpio y tareas de compresión mediante compress. El paquete resultante tiene un directorio adicional en el directorio superior llamado `archive`. En este directorio se encuentran todos los archivos nombrados por clase. El directorio `install` contendrá las secuencias de comandos de acción de clase necesarias para desempaquetar cada archivo. Las rutas absolutas no se archivan.

La utilidad `faspac` tiene el formato siguiente:

```
faspac [-m Archive Method] -a -s -q [-d Base Directory] /
[-x Exclude List] [List of Packages]
```

Cada opción del comando `faspac` se describe en la tabla siguiente.

Opción	Descripción
`-m` *Archive Method*	Indica un método para el archivado o la compresión. `bzip2` es la utilidad de compresión predeterminada que se ha utilizado. Para conmutar el método de compresión o descompresión, utilice `-m zip`; para cpio o compress use `- m cpio`.
`-a`	Corrige atributos (debe ser root para ello).
`-s`	Indica la traducción de paquetes de tipo ABI estándar. Esta opción toma un paquete comprimido o cpio y lo convierte en formato estándar de paquete compatible con ABI.
`-q`	Indica el modo silencioso.

Opción	Descripción
-d *Base Directory*	Indica el directorio donde se actuará sobre todos los paquetes presentes según lo requiera la línea de comandos. Esto es incompatible con la entrada *List of Packages*.
-x *Exclude List*	Indica una lista de paquetes separados por espacios, entre comillas o separados por comas para excluirlos del proceso.
List of Packages	Indica la lista de paquetes que se van a procesar.

Glosario

ABI Consulte Interfaz de aplicación binaria (ABI).

abreviatura de paquete Nombre corto de un paquete que se define mediante el parámetro `PKG` del archivo `pkginfo`.

almacén de claves del paquete Repositorio de certificados y claves que las herramientas de paquetes pueden consultar.

archivo `compver` Método para especificar la compatibilidad con versiones anteriores del paquete.

archivo de control Archivo que controla dónde, cómo y si un paquete se va a instalar. Consulte el archivo de información y la secuencia de comandos de instalación.

archivo de información Archivo que puede definir dependencias de paquetes, ofrecer un mensaje de copyright o reservar espacio adicional en un sistema de destino.

archivo `depend` Método para resolver dependencias de paquetes básicas. Consulte también el archivo `compver`.

ASN.1 Consulte Notación de sintaxis abstracta 1 (ASN.1)

certificado de confianza Certificado que contiene un único certificado de clave pública que pertenece a otra entidad. Los certificados de confianza se usan en la verificación de firmas digitales y al iniciar una conexión con un servidor seguro (SSL).

certificate authority (entidad emisora de certificados) Agencia, como Verisign, que emite certificados utilizados en la firma de paquetes.

clase Nombre que se usa para agrupar objetos de paquetes. Consulte también Secuencia de comandos de acción de clase.

clave de usuario Clave que contiene información de clave criptográfica sensible. Esta información se almacena con un formato protegido para impedir el uso no autorizado. Las claves de usuarios se usan cuando se crea un paquete firmado.

clave privada Una clave de cifrado o descifrado conocido solamente por las partes que intercambian mensajes secretos. Esta clave privada se usa junto con claves públicas para crear paquetes firmados.

clave pública Valor generado como clave de cifrado que, en combinación con la clave privada derivada de la clave pública, se puede usar para cifrar de forma eficaz los mensajes y las firmas digitales.

copyright Derecho de poseer y vender propiedad intelectual, como software, código fuente o documentación. La propiedad debe indicarse en el CD-ROM y en el texto de inserción, independientemente de si la propiedad intelectual es de SunSoft o de terceros. La propiedad del copyright también se reconoce en la documentación de SunSoft.

dependencia inversa Una condición cuando otro paquete depende de la existencia de su paquete. Consulte también archivo `depend`.

DER Consulte las normas de codificación distinguidas.

directorio base Ubicación donde se instalarán los objetos reubicables. Se define en el archivo `pkginfo` mediante el parámetro `BASEDIR`.

estándar de criptografía de clave pública n.º 12 Estándar que describe una sintaxis para almacenar objetos criptográficos en el disco. El almacén de claves de paquetes se mantiene en este formato.

estándar de criptografía de clave pública n.º 7 Estándar que describe una sintaxis general para datos a los que se ha aplicado criptografía, como firmas y sobres digitales. Un paquete firmado contiene una firma PKCS7 incorporada.

firma digital Mensaje codificado utilizado para verificar la integridad y seguridad de un paquete.

identificador de paquete Sufijo numérico que se agrega a una abreviatura de paquete mediante el comando `pkgadd`.

instancia de paquete Variación de un paquete que se determina mediante la combinación de definiciones de los parámetros `PKG`, `ARCH` y `VERSION` en el archivo `pkginfo` del paquete.

interfaz binaria de aplicaciones Definición de la interfaz del sistema binario entre aplicaciones compiladas y el sistema operativo en el que se ejecutan.

lista de parches Lista de parches que afectan al paquete actual. Esta lista de parches se registra en el paquete instalado del archivo `pkginfo`.

mensaje de privacidad mejorada Modo de codificar un archivo mediante codificación 64 base y algunos encabezados optativos. Se usa ampliamente para codificar certificados y claves privadas en un archivo que existe en un sistema de archivos o en un mensaje de correo electrónico.

momento de instalación Tiempo durante el que un paquete se instala con el comando `pkgadd`.

nombre común Nombre de alias enumerado en el almacén de claves del paquete para paquetes firmados.

nombre de ruta paramétrico Nombre de ruta que incluye la especificación de una variable.

normas de codificación distinguidas Representación binaria de un objeto ASN.1 y define cómo un objeto ASN.1 se serializa para el almacenamiento o la transmisión en los entornos computacionales. Se utiliza con paquetes firmados.

notación de sintaxis abstracta 1 Modo de expresar objetos abstractos. Por ejemplo, ASN.1 define un certificado de clave pública, todos los objetos que componen el certificado, y el orden en el que los objetos se recogen. Sin embargo, ASN.1 no especifica cómo los objetos se serializan para el almacenamiento o la transmisión.

objeto de paquete Otro nombre para un archivo de aplicación que se incluye en un paquete que se instalará en un sistema de destino.

objeto reubicable Objeto de paquete que no necesita una ubicación de ruta absoluta en un sistema de destino. En su lugar, su ubicación viene determinada durante el proceso de instalación. Consulte también Objeto reubicable colectivamente y Objeto reubicable individualmente.

objeto reubicable colectivamente Objeto de paquete que se sitúa en relación a una base de instalación común. Consulte también Directorio base.

objeto reubicable individualmente Objeto de paquete que no se restringe a la misma ubicación de directorio que un objeto reubicable colectivamente. Se define mediante una variable de instalación en el campo *path* del archivo `prototype`, y la ubicación de instalación se determina mediante una secuencia de comandos `request` o `checkinstall`.

paquete Colección de archivos y directorios necesarios para una aplicación de software.

paquete compuesto Paquete que contiene nombres de rutas reubicables y absolutas.

paquete de prerrequisito Paquete que depende de la existencia de otro paquete. Consulte también archivo `depend`.

paquete incompatible Paquete que es incompatible con el paquete nombrado. Consulte también archivo `depend`.

paquete sin firmar Paquete ABI normal, sin cifrado o firmas digitales.

paquetes firmados Paquete con formato de flujo normal con una firma digital que verifica lo siguiente: que el paquete procede de la entidad que lo firmó, la entidad que lo firmó, que el paquete no se ha modificado desde que la entidad lo firmó y que la entidad que lo firmó es de confianza.

PEM Consulte Mensaje de privacidad mejorada.

PKCS12 Consulte Criptografía de clave pública estándar n.º 12.

PKCS7 Consulte Criptografía de clave pública estándar n.º 7.

Recomendación ITU-T X.509 Protocolo que especifica la sintaxis del certificado de clave pública X.509 ampliamente adoptado.

reubicable Objeto de paquete definido en un archivo `prototype` con un nombre de ruta relativa.

secuencia de comandos de acción de clase Archivo que define un conjunto de acciones que se deben llevar a cabo en un grupo de objetos de paquetes.

secuencia de comandos de instalación Secuencia de comandos que permite proporcionar procedimientos de instalación personalizada para un paquete.

secuencia de comandos de procedimientos Secuencia de comandos que define acciones que tienen lugar en un momento determinado durante la instalación y la eliminación de paquetes.

segmentado Paquete que no se ajusta a un único volumen, como un disquete.

`tar` Recuperación de archivo de cinta. Comando de Oracle Solaris para agregar o extraer archivos de un medio.

tiempo de creación Momento durante el que un paquete se construye con el comando `pkgmk`.

variable de creación Variable que comienza por minúscula y se evalúa en el momento de la construcción.

variable de instalación Variable que comienza por mayúsculas y se evalúa en el tiempo de la instalación.

X.509 Consulte Recomendación ITU-T X.509.

Índice

B

C

R

S

T

U

V