Guía del nuevo desarrollador de Debian

Josip Rodin <joy-mg@debian.org>
Traducido por: Javier Fernández-Sanguino Peña <jfs@debian.org>
Traducido por: David Martínez <ender@debian.org>
Traducido por: Ana Beatriz Guerrero López <ana@debian.org>

version 1.2.11, 12 de enero de 2007.

Nota de Copyright

Índice general

Capítulo 1

Empezando «de la forma correcta».

Este documento tratará de describir cómo se construye un paquete Debian GNU/Linux para el usuario común de Debian y para futuros desarrolladores en un lenguaje informal, y con multitud de ejemplos. Hay un antiguo dicho romano que dice, *Longum iter est per preaecepta, breve et efficax per exempla!* (¡Es un largo camino con las reglas, pero corto y eficiente con ejemplos!)

Una de las cosas que hace a Debian una de las distribuciones más importantes del mercado es su sistema de paquetes. Aunque hay una gran cantidad de programas disponibles en forma de paquetes de Debian, algunas veces necesitarás instalar programas que no están disponible en este formato. Puede que te preguntes cómo hacer tus propios paquetes y que pienses que quizás ésta es una tarea demasiado difícil. Bueno, si eres un principiante en Linux, sí es duro, pero si eres un novato, no deberías estar leyendo esto ahora mismo. :-) Necesitas saber algo sobre programación en Unix, pero, desde luego, no tienes que ser un maestro.

Sin embargo, hay una cosa que es verdad: para crear y mantener paquetes Debian adecuadamente, necesitarás muchas horas. Para que nuestro sistema trabaje sin errores, nuestros desarrolladores necesitan ser técnicamente competentes y concienzudos.

Este documento explicará cada pequeño paso (al principio quizás irrelevantes), te ayudará a crear tu primer paquete, ganar alguna experiencia en construir próximas versiones de él, y quizás otros paquetes después.

Se pueden obtener versiones nuevas de este documento en línea en `http://www.debian.org/doc/maint-guide/` y en el paquete «maint-guide-es».

1.1. Programas que necesitas para el desarrollo

Antes de empezar nada, deberías asegurarte de que tienes instalados algunos paquetes adicionales necesarios para el desarrollo. Observa que en la lista no están incluidos paquetes cuyas prioridades son «esencial» o «requerido», que se suponen ya instalados.

En la revisión de este documento se han actualizado los paquetes para Debian 2.2 («potato») y 3.0 («woody»).

Los siguientes paquetes vienen en una instalación estándar de Debian, así que probablemente ya los tengas (junto con los paquetes de los que dependen). Aún así, deberías comprobarlo con «dpkg -s <paquete>».

- `dpkg-dev` - este paquete contiene las herramientas necesarias para desempaquetar, construir y enviar paquetes fuente de Debian (véase `dpkg-source(1)`).
- `file` - este útil programa puede determinar de qué tipo es un fichero (véase `file(1)`).
- `gcc` - el compilador de C de GNU, necesario si el programa, como la gran mayoría, está escrito en el lenguaje de programación C (véase `gcc(1)`). Este paquete también vendrá con otros paquetes como `binutils` que incluye programas para ensamblar y enlazar ficheros objeto (véase «info binutils» en el paquete `binutils-doc`) y el preprocesador de C en el paquete `cpp` (véase `cpp(1)`).
- `libc6-dev` - las bibliotecas y archivos de cabecera de C que gcc necesita para enlazar y crear ficheros objeto (véase «info libc» en el paquete `glibc-doc`).
- `make` - habitualmente la creación de un programa consta de varios pasos. En lugar de ejecutar las mismas órdenes una y otra vez, puedes utilizar este programa para automatizar el proceso, creando ficheros «Makefile» (véase «info make»).
- `patch` - esta utilidad es muy práctica, ya que permite tomar un fichero que contiene un listado de diferencias (producido por el programa diff) y aplicárselas al fichero original, produciendo una versión "parcheada". (véase `patch(1)`)
- `perl` - Perl es uno de los lenguajes interpretados para hacer guiones (o «scripts») más usados en los sistemas Un*x de hoy en dia, comúnmente se refiere a él como la «navaja suiza de Unix» (véase `perl(1)`).

Probablemente, necesitarás instalar además los siguientes paquetes:

- `autoconf` y `automake` - muchos programas nuevos usan ficheros de configuración y ficheros «Makefile» que se procesan con la ayuda de programas como éstos (véase «info autoconf», «info automake»).
- `dh-make` y `debhelper` - dh-make es necesario para crear el esqueleto de nuestro paquete ejemplo, y se usarán algunas de las herramientas de debhelper para crear los paquetes. Aunque no son imprescindibles para la creación de paquetes se recomiendan **encarecidamente** para nuevos desarrolladores. Hacen el proceso mucho más fácil al principio, y más fácil de controlar también más adelante (véase `dh_make(1)`, `debhelper(1)`, /usr/share/doc/debhelper/README).
- `devscripts` - este paquete contiene algunos guiones útiles para los desarrolladores, pero no son necesarios para crear paquetes (véase /usr/share/doc/devscripts/README.gz).

- `fakeroot` - esta utilidad te permite emular al usuario administrador (o «root», N. del T.), lo cual es necesario para ciertas partes del proceso de construcción (véase `fakeroot(1)`).

- `gnupg` - herramienta que te permite *firmar* digitalmente los paquetes. Esto es especialmente importante si quieres distribuir tu paquete a otras personas, y ciertamente, tendrás que hacerlo cuando tu trabajo vaya a incluirse en la distribución de Debian (véase `gpg(1)`).

- `g77` - el compilador GNU de Fortran 77, necesario si el programa está escrito en Fortran (véase `g77(1)`).

- `gpc` - el compilador GNU de Pascal, necesario si el programa está escrito en Pascal. Merece la pena mencionar aquí `fp-compiler`, un compilador libre de Pascal, que también es bueno en esta tarea (véase `gpc(1)`, `ppc386(1)`).

- `xutils` - algunos programas, normalmente aquellos hechos para X11, también usan programas para generar Makefiles de un conjunto de funciones de macro (véase `imake(1)`, `xmkmf(1)`).

- `lintian` - este es el comprobador de paquetes de Debian, que te indica muchos de los errores comunes después de construir un paquete, y explica los errores encontrados (véase `lintian(1)`, /usr/share/doc/lintian/lintian.html/index.html).

- `linda` - este es un comprobador de paquetes de Debian alternativo (véase `linda(1)`).

- `pbuilder` - este paquete contiene programas para crear y mantener entornos chroot. Al construir paquetes Debian en estos entornos chroot se verifica que las dependencias son las adecuadas y se evitan fallos al construir desde el código fuente (véase `pbuilder(8)` y `pdebuild(1)`).

Por último, la documentación que se indica a continuación es de *gran importancia* y debería leerse junto con este documento:

- `debian-policy` - incluye la estructura y contenidos del archivo, ciertas notas sobre el diseño del sistema operativo, el estándar de la jerarquía del sistema de ficheros («Filesystem Hierarchy Standard», N. del T.), y, lo más importante para ti, describe los requisitos que debe satisfacer cada paquete para ser incluido en la distribución (véase /usr/share/doc/debian-policy/policy.html/index.html).

- `developers-reference` - para todos los temas no específicamente relacionados con los detalles técnicos de cómo empaquetar, tales como la estructura del archivo, cómo renombrar, abandonar, adoptar paquetes, cómo hacer NMUs («Non-Maintainer Uploads», o envíos por personas distintas del desarrollador, N. del T.), como gestionar los errores que los usuarios envían, buenas prácticas de empaquetado, cómo y cuando enviar los paquetes, etc. (véase /usr/share/doc/developers-reference/index.en-us.iso-8859-1.html).

Las breves descripciones dadas anteriormente sólo sirven para introducirte a lo que hace cada paquete. Antes de continuar, por favor, lee la documentación de cada programa, al menos para su uso normal. Puede parecerte algo duro ahora, pero más adelante estarás *muy* contento de haberla leído.

Nota: `debmake` es un paquete que incluye otros programas con funciones similares a dh-make, pero su uso específico **no** está cubierto en este documento porque se trata de una herramienta obsoleta.

1.2. Desarrollador oficial de Debian

Puede que quieras convertir en un desarrollador oficial de Debian una vez hayas construido tu paquete (o incluso mientras lo estás haciendo) para que el paquete se introduzca en la nueva distribución (si el programa es útil, ¿por qué no?).

No puedes convertirte en desarrollador oficial de Debian de la noche a la mañana porque hace falta más que sólo habilidades técnicas. No te sientas desilusionado por esto. Aún puedes subir tu paquete, si es útil a otras personas, como su mantenedor a través de un patrocinador mientras tu entras en el proceso de nuevos desarrolladores de Debian (`http://nm.debian.org/`). En este caso el patrocinador es un desarrollador oficial de Debian que ayuda a la persona que mantiene el paquete a subirlo al archivo de Debian. Encontrarás más información de este procedimiento en el documento preguntas frecuentes de debian-mentors (`http://people.debian.org/~mpalmer/debian-mentors_FAQ.html`).

Ten en cuenta que no tienes que crear un paquete nuevo para poder convertirte en desarrollador oficial de Debian. Un camino posible para ser desarrollador oficial es contribuir al mantenimiento de los paquetes ya existentes en la distribución.

1.3. Más información

Puedes construir dos tipos de paquetes: fuentes y binarios. Un paquete fuente contiene el código que puedes compilar en un programa. Un paquete binario contiene sólo el programa terminado. ¡No mezcles los términos como «fuentes de un programa» y el «paquete fuente de un programa»! Por favor, lee los otros manuales si necesitas más detalles sobre terminología.

Debian usa el término desarrollador («maintainer», N. del T.) para la persona que hace paquetes, autor original («upstream author», N. del T.) para la persona que hizo el programa, y desarrollador original («upstream maintainer», N. del T.) para la persona que actualmente mantiene el programa fuera de Debian. Generalmente el autor y el desarrollador fuente son la misma persona - y algunas veces incluso el desarrollador es el mismo. Si haces un programa, y quieres incluirlo en Debian, tienes total libertad para solicitar convertirte en desarrollador.

Capítulo 2

Primeros pasos

2.1. Elige el programa

Probablemente hayas escogido ya el paquete que deseas construir. Lo primero que debes hacer es comprobar si el paquete está ya en el archivo de la distribución utilizando `aptitude`. Si usas la distribución «estable», quizás sea mejor que vayas a la página de búsqueda de paquetes (`http://www.debian.org/distrib/packages`).

Si el paquete ya existe, ¡instálalo! :-) Si te encuentras con que el paquete es un paquete huérfano (cuando su desarrollador es el «Debian QA Group», es decir, el grupo de calidad de Debian), entonces podrías adoptarlo.

Consulta la lista de paquetes en prospección y que necesitan trabajo (`http://www.debian.org/devel/wnpp/`) así como las páginas enlazadas a ésta para verificar si el paquete está huérfano o está siendo adoptado.

Si puedes adoptar el paquete, descarga las fuentes (con algo como `apt-get source packagename`) y examínalas. Este documento, desafortunadamente, no incluye aún información exhaustiva sobre la adopción de paquetes. No debería ser díficil entender cómo funciona el paquete ya que alguien ha hecho el trabajo inicial por ti. Aún así es mejor que sigas leyendo, muchos de los consejos que se dan a continuación serán también aplicables para tu caso.

Si el paquete es nuevo y decides que te gustaría verlo en Debian debes seguir los pasos indicados a continuación:

- Comprueba que no hay nadie más trabajando ya en el paquete consultando la lista de paquetes en los que se está trabajando (`http://www.de.debian.org/devel/wnpp/being_packaged`). Si ya hay alguien trabajando en él, contacta con esa persona. Si no, intenta encontrar otro programa interesante que nadie mantenga.
- El programa **debe** tener una licencia. Preferiblemente la licencia deberá ser libre en el sentido marcado por las Directrices de Debian para el software libre (`http://www.debian.org/social_contract.html#guidelines`) y **no puede** depender de un

paquete que no esté dentro de «main» para compilarse o para poder utilizarse. Si la licencia no sigue alguna de estas reglas aún puede incluirse en las secciones «contrib» o «non-free» de Debian dependiendo de su situación. Si no estás seguro sobre en qué lugar debería ir, envía el texto de la licencia y pide consejo con un correo (en inglés) dirigido a `<debian-legal@lists.debian.org>`.

- El programa **no** debería ejecutarse con «setuid root», o aún mejor: no debería ser «setuid» ni «setgid».
- El programa no debería ser un demonio, o algo que vaya en los directorios */sbin, o abrir un puerto como usuario administrador.
- El programa debería estar compuesto por binarios ejecutables, no intentes empaquetar aún con bibliotecas.
- El programa debería tener una buena documentación, o al menos un código fuente legible y no ofuscado.
- Deberías contactar con el autor o autores del programa para comprobar si está/n de acuerdo con que se empaquete. Es importante que el autor o autores sigan manteniendo el programa para que puedas en el futuro consultarle/s en caso de que haya problemas específicos. No deberías intentar empaquetar programas que no estén mantenidos.
- Y por último, pero no menos importante, deberías saber cómo funciona, y haberlo utilizado durante algún tiempo.

Por supuesto, esta lista es para tomar medidas de seguridad, y con la intención de salvarte de usuarios enfurecidos si haces algo mal con algún demonio «setuid»... Cuando tengas más experiencia en empaquetar, podrás hacer este tipo de paquetes, incluso los desarrolladores más experimentados preguntan en la lista de correo de debian-devel cuando tienen dudas. La gente allí te ayudará gustosamente.

Para más ayuda sobre esto, lee la Referencia del desarrollador.

2.2. Obtén el programa y pruébalo

Lo primero que debes hacer es encontrar y descargar el paquete original. A partir de este punto se da por supuesto que ya tienes el código fuente que obtuviste de la página del autor. Las fuentes de los programas libres de Linux generalmente vienen en formato tar/gzip, con extensión .tar.gz, y generalmente contienen un subdirectorio llamado «programa-versión» con todas las fuentes en él. Si tu programa viene en otro tipo de archivo (por ejemplo, el fichero termina en ".Z" o ".zip"), descomprímelo con las herramientas adecuadas, o pregunta en la lista de correo debian-mentors si tienes dudas de cómo se puede desempaquetar correctamente (pista: prueba «file archivo.extensión»).

Como ejemplo, usaré el programa conocido como «gentoo», un gestor de ficheros de X11 en GTK+. Observa que el programa ya ha sido empaquetado previamente pero ha cambiado sustancialmente de versión desde que este texto se escribió.

Crea un subdirectorio bajo tu directorio personal llamado «debian» o «deb» o lo que creas apropiado (por ejemplo `~/gentoo/` estaría bien en este caso). Mueve a él el archivo que has descargado, y descomprímelo de la siguiente forma: «tar xzf gentoo-0.9.12.tar.gz». Asegúrate de que no hay errores, incluso errores «irrelevantes», porque es muy probable que haya problemas al desempaquetarlo en sistemas de otras personas, cuyas herramientas de desempaquetado puede que no ignoren estas anomalías.

Ahora tienes otro subdirectorio, llamado «gentoo-0.9.12». Muévete a ese directorio y lee **en profundidad** la documentación que encuentres. Generalmente se encuentra en ficheros que se llaman README*, INSTALL*, *.lsm o *.html. Allí encontrarás instrucciones de cómo compilar e instalar el programa (muy probablemente asumirán que lo quieres instalar en el directorio /usr/local/bin, no harás esto, pero eso lo veremos más adelante en 'Instalación en un subdirectorio' en la página 9).

El proceso varía de un programa a otro, pero gran parte de los programas modernos vienen con un guión «configure» que configura las fuentes para tu sistema y se asegura de que el sistema está en condiciones de compilarlo. Después de configurarlo (con «./configure»), los programas generalmente se compilan con «make». Algunos de ellos soportan «make check» para ejecutarse incluyendo comprobaciones automáticas. Generalmente se instalarán en sus directorios de destino ejecutando «make install».

Ahora intenta compilar, y ejecutar el programa, para asegurarte de que funciona bien y de que no rompe nada mientras está instalándose o ejecutándose.

También, generalmente, puedes ejecutar «make clean» (o mejor «make distclean») para limpiar el directorio donde se genera el programa. A veces hay incluso un «make uninstall» que se puede utilizar para borrar todos los archivos instalados.

2.3. Nombre del paquete y versión

Deberías empezar a construir tu paquete en un directorio de fuentes completamente limpio, o simplemente con las fuentes recién desempaquetadas.

Para construir correctamente el paquete, debes cambiar el nombre original del programa a letras minúsculas (si no lo está ya), y deberías renombrar el directorio de fuentes a <nombre_de_paquete>-<versión>.

Si el nombre del programa está formado por varias palabras, contráelas a una palabra o haz una abreviatura. Por ejemplo, el paquete del programa «el editor para X de Javi» se podría llamar javiedx o jle4x, o lo que decidas, siempre y cuando no se exceda de unos límites razonables, como 20 caracteres.

Comprueba también la versión exacta del programa (la que se incluye en la versión del paquete). Si el programa no está numerado con versiones del estilo de X.Y.Z, pero sí con fecha de publicación, eres libre de utilizar la fecha como número de versión, precedida por «0.0» (sólo por si los desarrolladores originales deciden sacar una versión nueva como 1.0). Así, si la fecha de las fuentes es el 19 de diciembre de 1998, puedes utilizar la cadena 0 0.0.19981219 (que utiliza el formato de fecha norteamericano, N. del T.) como número de versión.

Aún así habrá algunos programas que ni siquiera estén numerados, en cuyo caso deberás contactar con el desarrollador original para ver si tienen algún otro sistema de seguimiento de revisiones.

2.4. «Debianización» inicial

Asegúrate que te encuentras en el directorio donde están las fuentes del programa y ejecuta lo siguiente:

```
dh_make -e tu.dirección@de.desarrollador -f ../gentoo-0.9.12.tar.gz
```

Por supuesto, cambia la cadena «tu.dirección@de.desarrollador» por tu dirección de correo electrónico para que se incluya en la entrada del fichero de cambios así como en otros ficheros, y el nombre de fichero de tu archivo fuente original. Lee `dh_make(1)` para más detalles.

Saldrá alguna información. Te preguntará qué tipo de paquete deseas crear. Gentoo es un sólo paquete de binarios - crea sólo un binario, y, por tanto, sólo un fichero .deb - así que seleccionaremos la primera opción, con la tecla «s». Comprueba la información que aparece en la pantalla y confirma pulsando la tecla <intro>.

Tras ejecutar `dh_make`, se crea una copia del código original con el nombre `gentoo_0.9.12.orig.tar.gz` en el directorio raíz para facilitar la creación del paquete de fuentes no nativo de Debian con el `diff.gz`. Observa que hay dos cambios clave en este nombre de fichero:

- El nombre del paquete y la versión están separados por «_».
- Hay un «`orig.`» antes de «`tar.gz`».

Como nuevo desarrollador, se desaconseja crear paquetes complicados, por ejemplo:

- múltiples paquetes binarios
- paquetes de bibliotecas
- paquetes en los que el formato del archivo fuente no es en `tar.gz.` ni en `tar.bz2`, o
- paquetes cuyas fuentes contienen partes que no se pueden distribuir.

Estos casos no son extremadamente difíciles, pero sí necesita algunos conocimientos más, así que aquí no se describirá el proceso de empaquetado para este tipo de paquetes.

Ten en cuenta que deberías ejecutar `dh_make` **sólo una vez**, y que no se comportará correctamente si lo haces otra vez en el mismo directorio ya «debianizado». Esto también significa que usarás un método distinto para crear una nueva revisión o una nueva versión de tu paquete en el futuro. Lee más sobre esto más adelante, en 'Actualizar el paquete' en la página 43.

Capítulo 3

Modificar las fuentes

Por lo general, los programas se instalan a sí mismos en el subdirectorio `/usr/local`. Pero los paquetes Debian no pueden utilizar este directorio ya que está reservado para el uso privado del administrador (o de los usuarios). Esto significa que tienes que mirar el sistema de construcción de tu programa, generalmente empezando por el fichero «Makefile». Éste es el guión `make(1)` que se usará para automatizar la creación de este programa. Para más detalles sobre los ficheros «Makefiles», consulta 'El fichero «rules»' en la página 20.

Observa que si tu programa usa GNU `automake(1)` y/o `autoconf(1)`, lo que quiere decir que las fuentes incluyen ficheros Makefile.am y Makefile.in, respectivamente, ya que necesitarás modificar esos ficheros, porque cada invocación de automake reescribirá los ficheros «Makefile.in» con información generada a partir de los ficheros «Makefile.am», y cada llamada a `./configure` hará lo mismo con los ficheros «Makefile», con información de los ficheros «Makefile.in». Editar los ficheros «Makefile.am» requiere algunos conocimientos de `automake`, que puedes obtener leyendo la entrada de `info` para automake, mientras que editar los ficheros «Makefile.in» es casi lo mismo que editar ficheros «Makefile», simplemente basta con poner atención en las variables, es decir, cualquier cadena que empiece y acabe con el caracter «@», como por ejemplo @CFLAGS@ o @LN_S@, que se sustituyen por otros valores cada vez que se ejecute `./configure`. Por favor, lee `/usr/share/doc/autotools-dev/README.Debian.gz` antes de empezar.

Ten en cuenta que no hay espacio aquí para entrar en *todos* los detalles respecto a los arreglos que deben hacerse en las fuentes originales. Sin embargo, a continuación se detallan algunos de los problemas más frecuentes.

3.1. Instalación en un subdirectorio

La mayor parte de los programas tienen alguna manera de instalarse en la estructura de directorios existente en tu sistema, para que los binarios sean incluidos en tu $PATH, y para que encuentre la documentación y páginas de manual en los lugares habituales. Sin embargo, si lo instalas de esta forma, el programa se instalará con los demás binarios que ya están en tu

sistema. Esto dificultará a las herramientas de paquetes averiguar qué archivos pertenecen a tu paquete y cuales no.

Por lo tanto, necesitas hacer algo más: instalar el programa en un subdirectorio temporal desde el cual las herramientas de desarrollo construirán el paquete .deb que se pueda instalar. Todo lo que se incluye en este directorio será instalado en el sistema del usuario cuando instale su paquete, la única diferencia es que dpkg instalará los ficheros en el directorio raíz.

Este directorio temporal se creará bajo el directorio debian/ que está dentro del árbol del código descomprimido, generalmente con el nombre `debian/nombre_de_paquete`.

Ten en cuenta que, aunque necesitas que el programa se instale en `debian /nombre_de_paquete`, también necesitas que se comporte correctamente cuando se instale en el directorio raíz, es decir, cuando se instale desde el paquete .deb. Así que no deberías permitir que al construirse lo haga con cadenas como `/home/me/deb/gentoo-0.9.12/usr/share/gentoo` dentro de los archivos del paquete a distribuir.

Esto será sencillo con los de programas que utilicen la herramienta GNU `autoconf`. La mayoría de estos programas tienen ficheros «Makefile» por omisión que permiten configurar la instalación en un subdirectorio cualquiera, aunque recordando que, por ejemplo, `/usr` es el prefijo normal. Cuando detecte que tu programa usa `autoconf`, `dh_make` fijará las opciones necesarias para hacer esto automáticamente, así que puedes dejar de leer esta sección. Pero con otros programas puede ser necesario que examines y edites los ficheros «Makefile».

Esta es la parte importante del Makefile de gentoo:

```
# ¿Dónde poner el binario en «make install»?
BIN     = /usr/local/bin

# ¿Dónde poner los iconos en «make install»?
ICONS   = /usr/local/share/gentoo/
```

Vemos que los ficheros están configurados para instalarse bajo `/usr/local`. Cambia estas rutas a:

```
# ¿Dónde poner el binario en «make install»?
BIN     = $(DESTDIR)/usr/bin

# ¿Dónde poner los iconos en «make install»?
ICONS   = $(DESTDIR)/usr/share/gentoo
```

Pero: ¿por qué en este directorio y no en otro? Porque los paquetes de Debian nunca se instalan bajo `/usr/local`, este árbol de directorio, está reservado para el uso del administrador del sistema. Así que estos ficheros deben instalarse en `/usr`.

La localización correcta de los binarios, iconos, documentación, etc, está especificada en el «Estándar de la jerarquía del sistema de ficheros» (véase /usr/share/doc/debian-policy/fhs). Te recomiendo que leas las secciones que podrían aplicar a tu paquete.

Así pues, deberíamos instalar el binario en /usr/bin en lugar de /usr/local/bin y la página de manual en /usr/share/man/man1 en lugar de /usr/local/man/man1. No hemos mencionado ninguna página de manual en el Makefile de gentoo, pero en Debian se requiere que cada programa debe tener una, así que haremos una más tarde y la instalaremos en /usr/share/man/man1.

Algunos programas no usan variables en el makefile para definir rutas como éstas. Esto significa que tendrás que editar algunos de los ficheros de código C para arreglarlos y que usen las rutas correctas. Pero, ¿dónde buscar?, y exactamente, ¿el qué? Puedes probar a encontrarlos usando:

```
grep -nr -e 'usr/local/lib' --include='*.[c|h]' .
```

(En cada subdirectorio que contenga ficheros .c y .h, `grep` nos indicará el nombre del fichero y la línea cuando encuentre una ocurrencia.

Ahora edita esos ficheros y cambia en esas líneas usr/local/lib con usr/share y ya está. Sólo tienes que reemplazar usr/local/lib por tu localización, pero debes ser muy cuidadoso para no modificar el resto del código, especialmente si no sabes mucho sobre cómo programar en C. :-)

Después de esto deberías encontrar el objetivo «install» (busca una línea que comience por «install:») y renombra todas las referencias a directorios distintos de los definidos al comienzo del Makefile. Anteriormente el objetivo «install» decía:

```
install:        gentoo
                install ./gentoo $(BIN)
                install icons $(ICONS)
                install gentoorc-example $(HOME)/.gentoorc
```

Después del cambio dice:

```
install:        gentoo-target
                install -d $(BIN) $(ICONS) $(DESTDIR)/etc
                install ./gentoo $(BIN)
                install -m644 icons/* $(ICONS)
                install -m644 gentoorc-example $(DESTDIR)/etc/gentoorc
```

Seguramente has notado que ahora hay una orden `install -d` antes de las demás órdenes de la regla. El makefile original no lo tenía porque normalmente /usr/local/bin y otros directorios ya existen en el sistema donde se ejecuta «make install». Sin embargo, dado que lo instalaremos en un directorio vacío (o incluso inexistente), tendremos que crear cada uno de estos directorios

También podemos añadir otras cosas al final de la regla, como la instalación de documentación adicional que los desarrolladores originales a veces omiten:

```
                install -d $(DESTDIR)/usr/share/doc/gentoo/html
                cp -a docs/* $(DESTDIR)/usr/share/doc/gentoo/html
```

Un lector atento se dará cuenta de que he cambiado «gentoo» a «gentoo-target» en la línea «install:». A eso se le llama arreglar un fallo en el programa. :-)

Siempre que hagas cambios que no estén específicamente relacionados con el paquete Debian, asegúrate de que los envías al desarrollador original para que éste los pueda incluir en la próxima revisión del programa y así le puedan ser útiles a alguien más. Además, recuerda hacer que tus cambios no sean específicos para Debian o Linux (¡ni siquiera para Unix!) antes de enviarlos, hazlo portable. Esto hará que tus arreglos sean más fáciles de aplicar.

Ten en cuenta que no tienes que enviar ninguno de los ficheros debian/* al desarrollador original.

3.2. Bibliotecas diferentes

Hay otro problema común: las bibliotecas son generalmente diferentes de plataforma a plataforma. Por ejemplo, un Makefile puede contener una referencia a una biblioteca que no exista en Debian o ni siquiera en Linux. En este caso, se necesita cambiarla a una biblioteca que sí exista en Debian y sirva para el mismo propósito.

Así, si hay una línea en el Makefile (o Makefile.in) de tu programa que dice algo como lo siguiente (y tu programa no compila):

```
LIBS = -lcurses -lcosas -lmáscosas
```

Entonces cámbiala a lo siguiente, y funcionará casi con seguridad:

```
LIBS = -lncurses -lcosas -lmáscosas
```

(El autor se ha dado cuenta de que éste no es el mejor ejemplo ya que ahora el paquete `libncurses` incluye un enlace simbólico a libcurses.so, pero no puedo pensar uno mejor. Cualquier sugerencia sería muy bien recibida :-)

Capítulo 4

Las cosas necesarias bajo debian/

Ahora hay un nuevo subdirectorio bajo el directorio principal del programa («gentoo-0.9.12»), que se llama «debian». Hay algunos ficheros en este directorio que debemos editar para adaptar el comportamiento del paquete. La parte más importante es modificar los ficheros «control», «rules», «changelog», y «copyright» que son necesarios en todos los paquetes.

4.1. El fichero «control»

Este fichero contiene varios valores que `dpkg`, `dselect` y otras herramientas de gestión de paquetes usarán para gestionar el paquete.

Aquí está el fichero de control que dh_make crea para nosotros:

```
1  Source: gentoo
2  Section: unknown
3  Priority: optional
4  Maintainer: Josip Rodin <joy-mg@debian.org>
5  Build-Depends: debhelper (» 3.0.0)
6  Standards-Version: 3.6.2
7
8  Package: gentoo
9  Architecture: any
10 Depends: ${shlibs:Depends}
11 Description: <insertar hasta 60 caracteres de descripción>
12  <inserta una descripción larga, indentada con espacios.>
```

(He añadido los números de línea).

Las líneas 1 a 6 son la información de control para el paquete fuente.

La línea 1 es el nombre del paquete fuente.

La línea 2 es la sección de la distribución dentro de la que estará este paquete.

Como puede que hayas notado, Debian está dividida en secciones: «main» (principal, N. del T.) (el software libre), «non-free» (no libre, N. del T.) (el software que realmente no es libre) y «contrib» (software libre que depende de software no libre). Bajo ellas hay subdivisiones lógicas que describen en una palabra qué paquetes hay dentro. Así que tenemos «admin» para programas que sólo usa un administrador, «base» para las herramientas básicas, «devel» para las herramientas de programación, «doc» para la documentación, «libs» para las bibliotecas, «mail» para lectores y demonios de correo-e, «net» para aplicaciones y demonios de red, «x11» para programas específicos de X11, y muchos más.

Vamos a cambiarla para que ponga x11. El prefijo "main/" ya va implícito, así que podemos omitirlo.

La línea 3 describe cómo de importante es para el usuario la instalación de este paquete. Podrás consultar en el manual de normas de Debian («Debian Policy», N. del T.) la guía de los valores que deberían tener estos campos. La prioridad «optional» suele ser lo mejor para los paquetes nuevos.

«Section» y «Priority» se usan en las interfaces como `dselect` cuando ordenan los paquetes. Una vez que envies el paquete a Debian, el valor de estos dos campos puede no ser aceptado por los responsables del archivo, en cuyo caso te lo notificarán por correo electrónico.

Como es un paquete de prioridad normal y no tiene conflictos con ningún otro, lo dejaremos con prioridad «optional» (opcional, N. del T.).

La línea 4 es el nombre y correo electrónico del desarrollador. Asegúrate de que este campo incluye una cabecera válida «To: », para una dirección de correo electrónico, porque despueé de que envíes el paquete, el sistema de seguimiento de errores («Bug Tracking System», N. del T.) utilizará esta dirección para enviarte los mensajes de los bugs. Evita usar comas, el signo «&» y paréntesis.

La línea 5 incluye la lista de paquetes requeridos para construir tu paquete. Algunos paquetes como gcc y make están implícitos, consulta el paquete `build-essential` para más detalles. Si se necesita algún compilador no estándar u otra herramienta para construir tu paquete, deberías añadirla en la línea «Build-Depends». Las entradas múltiples se separan con comas, lee la explicación de las dependencias binarias para averiguar más sobre la sintaxis de este campo.

También tienes los campos «Build-Depends-Indep» y «Build-Conflicts» entre otros. Estos datos los usarán los programas de construcción automática de paquetes de Debian para crear paquetes binarios para el resto de arquitecturas. Consulta las normas de Debian para más información sobre las dependencias de construcción y la Referencia del Desarrollador para más información sobre las otras arquitecturas y sobre cómo migrar los programas a ellas.

Aquí tienes un truco que puedes usar para averiguar qué paquetes necesitará tu paquete en su construcción:

```
strace -f -o /tmp/log ./configure
# o make en lugar de ./configure, si el paquete no usa autoconf
```

```
for x in ‘dpkg -S $(grep open /tmp/log|\
                    perl -pe ’s!.* open\(\"([^\"]*).*!$1!’ |\
                    grep "^/"| sort | uniq|\
                    grep -v "^\(/tmp\|/dev\|/proc\)" ) 2>/dev/null|\
                    cut -f1 -d":"| sort | uniq‘; \
      do \
        echo -n "$x (>=" ‘dpkg -s $x|grep ^Version|cut -f2 -d":"‘ "), "; \
      done
```

Para encontrar manualmente las dependencias exactas de */usr/bin/foo*, ejecuta

```
objdump -p /usr/bin/foo | grep NEEDED
```

y para cada biblioteca, por ejemplo, `libfoo.so.6`, ejecuta

```
dpkg -S libfoo.so.6
```

Debes utilizar la versión «-dev» de cada uno de los paquetes dentro de la entrada «Build-deps». Si usas `ldd` para este propósito, también te informará de las dependencias de bibliotecas indirectas, lo que puede llevar a que se introduzcan demasiadas dependencias de construcción.

Gentoo también requiere `xlibs-dev`, `libgtk1.2-dev` y `libglib1.2-dev` para su construcción, así que lo añadiremos junto a `debhelper`.

La línea 6 es la versión de los estándares definidos en las normas de Debian que sigue este paquete, es decir, la versión del manual de normas que has leído mientras haces tu paquete.

La línea 8 es el nombre del paquete binario. Este suele ser el mismo que el del paquete fuente, pero no tiene que ser necesariamente así siempre.

La línea 9 describe la arquitectura de CPU para la que el paquete binario puede ser compilado. Dejaremos puesto «any» (cualquiera, N. del T.), porque `dpkg-gencontrol(1)` la rellenará con el valor apropiado cuando se compile este paquete en cualquier arquitectura para la cual pueda ser compilado.

Si tu paquete es independiente de la arquitectura (por ejemplo, un documento, un guión escrito en Perl o para el intérprete de órdenes), cambia esto a «all», y consulta más adelante ‘El fichero «rules»’ en la página 20 sobre cómo usar la regla «binary-indep» en lugar de «binary-arch» para construir el paquete.

La línea 10 muestra una de las más poderosas posibilidades del sistema de paquetes de Debian. Los paquetes se pueden relacionar unos con otros de diversas formas. Aparte de «Depends:» (depende de, N. del T.) otros campos de relación son «Recommends:» (recomienda, N. del T.), «Suggests:» (sugiere, N. del T.), «Pre-Depends:» (predepende de, N. del T.), «Conflicts:» (entra en conflicto con, N. del T.), «Provides:» (provee, N. del T.), «Replaces:» (reemplaza a, N. del T.).

Las herramientas de gestión de paquetes se comportan habitualmente de la misma forma cuando tratan con esas relaciones entre paquetes; si no es así, se explicará en cada caso. (véase `dpkg(8)`, `dselect(8)`, `apt(8)`, `aptitude(1)`, etc.)

A continuación se detalla el significado de las dependencias:

- Depends:

  No se instalará el programa a menos que los paquetes de los que depende estén ya instalados. Usa esto si tu programa no funcionará de ninguna forma (o se romperá fácilmente) a no ser que se haya instalado un paquete determinado.

- Recommends:

  Programas como dselect o aptitude informarán en la instalación de los paquetes recomendados por tu paquete, dselect incluso insistirá. dpkg y apt-get ignorarán este campo. Usa esto para paquetes que no son estrictamente necesarios pero que se usan habitualmente con tu programa.

- Suggests:

  Cuando un usuario instale el paquete, todos los programas le informarán de que puede instalar los paquetes sugeridos. Salvo `dpkg` y `apt`, que ignorarán estas dependencias. Utiliza esto para paquetes que funcionarán bien con tu programa pero que no son necesarios en absoluto.

- Pre-Depends:

  Esto es más fuerte que «Depends». El paquete no se instalará a menos que los paquetes de los que pre-dependa esté instalados *y correctamente configurados*. Utiliza esto **muy** poco y sólo después de haberlo discutido en la lista de distribución de debian-devel. En resumidas cuentas: no lo utilices en absoluto :-)

- Conflicts:

  El paquete no se instalará hasta que todos los paquetes con los que entra en conflicto hayan sido eliminados. Utiliza esto si tu programa no funcionará en absoluto (o fallará fácilmente) si un paquete en concreto está instalado.

- Provides:

  Se han definido nombres virtuales para algunos tipos determinados de paquetes que ofrecen múltiples alternativas para la misma función. Puedes obtener la lista completa en el fichero /usr/share/doc/debian-policy/virtual-package-names-list.text.gz. Usa esto si tu programa ofrece las funciones de un paquete virtual que ya exista.

- Replaces:

  Usa esto si tu programa reemplaza ficheros de otro paquete o reemplaza totalmente otro paquete (generalmente se usa conjuntamente con «Conflicts:»). Se eliminarán los ficheros de los paquetes indicados antes de instalar el tuyo.

Todos estos campos tienen una sintaxis uniforme: se trata de una lista de nombres de paquetes separados por comas. Estos nombres de paquetes también puede ser listas de paquetes alternativos, separados por los símbolos de barra vertical | (símbolos tubería).

Los campos pueden restringir su aplicación a versiones determinadas de cada paquete nombrado. Esto se hace listando después de cada nombre de paquete individual las versiones entre paréntesis, e indicando antes del número de versión una relación de la siguiente lista. Las relaciones permitidas son: <<, <=, =, >= y >> para estrictamente anterior, anterior o igual, exactamente igual, posterior o igual o estrictamente posterior, respectivamente. Por ejemplo:

```
Depends: foo (>= 1.2), libbar1 (= 1.3.4)
Conflicts: baz
Recommends: libbaz4 (» 4.0.7)
Suggests: quux
Replaces: quux (« 5), quux-foo (<= 7.6)
```

La última funcionalidad que necesitas conocer es $(shlibs:Depends). Después de que tu paquete se compile y se instale en el directorio temporal, `dh_shlibdeps(1)` lo escaneará en busca de binarios y bibliotecas para determinar las dependencias de bibliotecas compartidas y en qué paquetes están, tales como como libc6 o xlib6g. Luego pasará la lista a `dh_gencontrol(1)` que rellenará estas dependencias en el lugar adecuado. De esta forma no tendrás que preocuparte por esto.

Después de decir todo esto, podemos dejar la línea de «Depends:» exactamente como está ahora e insertar otra línea tras ésta que diga `Suggests: file`, porque gentoo utiliza algunas funciones de este paquete/programa.

La línea 11 es una descripción corta. La mayor parte de los monitores de la gente son de 80 columnas de ancho, así que no debería tener más de 60 caracteres. Cambiaré esto a «fully GUI configurable GTK+ file manager» («Gestor de ficheros GTK+ completamente configurable por GUI»).

La línea 12 es donde va la descripción larga del paquete. Debería ser al menos un párrafo que dé más detalles del paquete. La primera columna de cada línea debería estar vacía. No puede haber líneas en blanco, pero puede poner un . (punto) en una columna para simularlo. Tampoco debe haber más de una línea en blanco después de la descripción completa.

Aquí está el fichero de control actualizado:

```
1  Source: gentoo
2  Section: x11
3  Priority: optional
4  Maintainer: Josip Rodin <joy-mg@debian.org>
5  Build-Depends: debhelper (» 3.0.0), xlibs-dev, libgtk1.2-dev, libglib1.2
6  Standards-Version: 3.5.2
7
8  Package: gentoo
```

```
9  Architecture: any
10 Depends: ${shlibs:Depends}
11 Suggests: file
12 Description: fully GUI configurable X file manager using GTK+
13  gentoo is a file manager for Linux written from scratch in pure C. It
14  uses the GTK+ toolkit for all of its interface needs. gentoo provides
15  100% GUI configurability; no need to edit config files by hand and re-
16  start the program. gentoo supports identifying the type of various
17  files (using extension, regular expressions, or the «file» command),
18  and can display files of different types with different colors and icon
19  .
20  gentoo borrows some of its look and feel from the classic Amiga file
21  manager "Directory OPUS" (written by Jonathan Potter).
```

(He añadido los números de línea).

4.2. El fichero «copyright»

Este fichero contiene la información sobre la licencia y copyright de las fuentes originales del paquete. El formato no está definido en las normas, pero sí en sus contenidos (sección 12.6 «Copyright information»).

dh_make crea por omisión un fichero como este:

```
1  This package was debianized by Josip Rodin <joy-mg@debian.org> on
2  Wed, 11 Nov 1998 21:02:14 +0100.
3
4  It was downloaded from <rellena con el sitio ftp site>
5
6  Upstream Author(s): <pon el nombre del autor y dirección de correo>
7
8  Copyright:
9
10 <Debe incluirse aquí>
```

(He añadido los números de línea).

Las cosas importantes que se deben añadir a este fichero son el lugar de donde obtuviste el paquete junto con la nota de copyright y licencia originales. Debes incluir la licencia completa, a menos que sea una licencia común en el mundo del software libre como GNU GPL o LGPL, BSD o la «Licencia artística», donde basta referirse al fichero apropiado en el directorio /usr/share/common-licenses/ que existe en todo sistema Debian.

Gentoo está publicado bajo la Licencia Pública General GNU, así que cambiaremos el fichero a esto:

```
1  This package was debianized by Josip Rodin <joy-mg@debian.org> on
2  Wed, 11 Nov 1998 21:02:14 +0100.
3
4  It was downloaded from: ftp://ftp.obsession.se/gentoo/
5
6  Upstream author: Emil Brink <emil@obsession.se>
7
8  This software is copyright (c) 1998-99 by Emil Brink, Obsession
9  Development.
10
11 You are free to distribute this software under the terms of
12 the GNU General Public License  either version 2 of the License,
13 or (at your option) any later version.
14 On Debian systems, the complete text of the GNU General Public
15 License can be found in the file '/usr/share/common-licenses/GPL-2'.
```

(He añadido los números de línea).

Por favor, sigue el COMO de debian-devel-announce: http://lists.debian.org/debian-devel-announce/2006/03/msg00023.html .

(Nota del T.: las normas actuales de Debian actuales indican que los documentos aquí citados estén escritos en inglés, al ser el idioma oficial del proyecto, por lo que no se traducen en este documento).

4.3. El fichero «changelog»

Este es un fichero requerido, que tiene un formato especial descrito en las normas, sección 4.4 "debian/changelog". Este es el formato que usan dpkg y otros programas para obtener el número de versión, revisión, distribución y urgencia de tu paquete.

Para ti es también importante, ya que es bueno tener documentados todos los cambios que hayas hecho. Esto ayudará a las personas que se descarguen tu paquete para ver si hay temas pendientes en el paquete que deberían conocer de forma inmediata. Se guardará como «/usr/share/doc/gentoo/changelog.Debian.gz» en el paquete binario.

dh_make crea uno por omisión, el cual es como sigue:

```
1  gentoo (0.9.12-1) unstable; urgency=low
2
3   * Initial Release.
4
5  -- Josip Rodin <joy-mg@debian.org>  Wed, 11 Nov 1998 21:02:14 +0100
6
```

(He añadido los números de línea).

La línea 1 es el nombre del paquete, versión, distribución y urgencia. El nombre debe coincidir con el nombre del paquete fuente, la distribución debería ser, por ahora, «unstable» (o incluso «experimental») y la urgencia no debería cambiarse a algo mayor que «low». :-)

Las línea 3-5 son una entrada de registro, donde se documentan los cambios hechos en esta revisión del paquete (no los cambios en las fuentes originales - hay un fichero especial para este propósito, creado por los autores originales y que instalarás luego como /usr/share/doc/gentoo/changelog.gz). Las nuevas líneas deben insertarse justo antes de la línea que hay más arriba que comienza por un asterisco («*»). Puede hacerlo con `dch(1)`, o manualmente con cualquier editor de texto.

Terminarás con algo así:

```
1  gentoo (0.9.12-1) unstable; urgency=low
2
3   * Initial Release.
4   * This is my first Debian package.
5   * Adjusted the Makefile to fix $DESTDIR problems.
6
7  -- Josip Rodin <joy-mg@debian.org> Wed, 11 Nov 1998 21:02:14 +0100
8
```

(He añadido los números de línea).

Puedes leer más sobre cómo actualizar el fichero changelog más adelante en ‘Actualizar el paquete’ en la página 43.

4.4. El fichero «rules»

Ahora necesitamos mirar las reglas exactas que `dpkg-buildpackage(1)` utilizará para crear el paquete. Este fichero es en realidad otro Makefile, pero diferente al que viene en las fuentes originales. A diferencia de otros ficheros en debian/, éste necesita ser un fichero ejecutable.

Cada fichero «rules» (de reglas, N. del T.), como muchos otros Makefiles, se compone de varias reglas que especifican cómo tratar las fuentes. Cada regla se compone de objetivos, ficheros o nombres de acciones que se deben llevar a cabo (por ejemplo, «build:» o «install:»). Las reglas que quieras ejecutar deberían llamarse como argumentos de la línea de órdenes (por ejemplo, «./debian/rules build» o «make -f rules install»). Después del nombre del objetivo, puedes nombrar las dependencias, programas o ficheros de los que la regla dependa. Después de esto, hay un número cualquiera de instrucciones (¡indentado con <tab>!), hasta que se llega a una línea en blanco. Ahí empieza otra regla. Las líneas múltiples en blanco, y las líneas que empiezan por almohadillas («#») se tratan como comentarios y se ignoran.

Probablemente ya te hayas perdido, pero todo quedará más claro después de ver el fichero «rules» que dh_make pone por omisión. Deberías leer también la entrada de «make» en info para más información.

La parte importante que debes conocer sobre el fichero de reglas creado por dh_make, es que sólo es una sugerencia. Funcionará para paquetes simples pero para más complicados, no te asustes y añade o quita cosas de éste para ajustarlo a tus necesidades. Una cosa que no debes cambiar son los nombres de las reglas, porque todas las herramientas utilizan estos nombres, como se describe en las normas.

Éste es, más o menos, el contenido del fichero debian/rules que dh_make genera por omisión:

```
1  #!/usr/bin/make -f
2  # Fichero debian/rules de ejemplo que utiliza debhelper.
3  # GNU copyright 1997 to 1999 by Joey Hess.
4
5  # Quítele el comentario para activar el modo de depuración
6  #export DH_VERBOSE=1
7
8  # Esta es la versión de compatibilidad con debhelper que usaremos.
9  export DH_COMPAT=4
10
11 CFLAGS = -g
12 ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
13 CFLAGS += -O0
14 else
15 CFLAGS += -O2
16 endif
17
18 build: build-stamp
18 build: build-stamp
19 build-stamp:
20        dh_testdir
21
22        # Añada aquí las órdenes para compilar el paquete.
23        $(MAKE)
24        #docbook-to-man debian/gentoo.sgml > gentoo.1
25
26        touch build-stamp
27
28 clean:
29        dh_testdir
30        dh_testroot
31        rm -f build-stamp
32
33        # Añada aquí las órdenes para limpiar después del proceso de crea
34        -$(MAKE) clean
35
36        dh_clean
37
```

```
38 install: build
39        dh_testdir
40        dh_testroot
41        dh_clean -k
42        dh_installdirs
43
44        # Añada aquí las órdenes para instalar el paquete en debian/gento
45        $(MAKE) install DESTDIR=$(CURDIR)/debian/gentoo
46
47 # Aquí van los ficheros que se compilan para todas las arquitecturas
48 binary-indep: build install
49 # Por omisión no se hace nada.
50
51 # Construir los ficheros dependientes de arquitectura aquí.
52 binary-arch: build install
53        dh_testdir
54        dh_testroot
55 #      dh_installdebconf
56        dh_installdocs
57        dh_installexamples
58        dh_installmenu
59 #      dh_installlogrotate
60 #      dh_installemacsen
61 #      dh_installpam
62 #      dh_installmime
63 #      dh_installinit
64        dh_installcron
65        dh_installman
66        dh_installinfo
67 #      dh_undocumented
68        dh_installchangelogs ChangeLog
69        dh_link
70        dh_strip
71        dh_compress
72        dh_fixperms
73 #      dh_makeshlibs
74        dh_installdeb
75 #      dh_perl
76        dh_shlibdeps
77        dh_gencontrol
78        dh_md5sums
79        dh_builddeb
80
81 binary: binary-indep binary-arch
82 .PHONY: build clean binary-indep binary-arch binary install
```

(He añadido los números de línea. En el fichero `debian/rules` los espacios iniciales de las líneas son códigos de tabulación)

(N. del T.: se han traducido los comentarios del fichero de reglas, pero en el fichero generado por dh_make estarán en inglés)

Probablemente estés familiarizado con líneas como la primera de guiones escritos en shell o Perl. Esta línea indica que el fichero debe ejecutarse con /usr/bin/make.

El significado de las variables DH_* que se mencionan en las líneas 6 y 9 debería ser evidente de la descripción corta. Para más información sobre DH_COMPAT consulte la sección «Debhelper compatibility levels» del manual de `debhelper(1)`.

Las líneas de la 11 a la 16 son el esqueleto de apoyo para los parámetros de DEB_BUILD_OPTIONS, descritos en las normas sección 10.1 «Binarios». Basicamente, estas cosas controlan si los binarios se construyen con los símbolos del depurador y si deberían eliminarse tras la instalación. De nuevo, es sólo un esqueleto, una pista de lo que deberías hacer. Deberías comprobar cómo el sistema de construcción de las fuentes maneja la inclusión de los símbolos del depurador y su eliminación en la instalación e implementarlo por ti mismo.

Habitualmente puedes decirle a gcc que compile con "-g" usando la variable CFLAGS. Si este es el caso de tu paquete, pon la variable *añadiendo* `CFLAGS="$(CFLAGS)"` a la invocación de $(MAKE) en la regla de construcción (ver más abajo). Alternativamente, si tu paquete usa un guión de configuración de autoconf puedes definir la cadena arriba mostrada *anteponiéndola* a la llamada a ./configure en la regla de construcción.

Los programas a los que se le quitan los símbolos del depurador con `strip` se configuran normalmente para instalarse sin pasar por `strip`, y a menudo sin una opción para cambiar esto. Afortunadamente, tienes `dh_strip(1)` que detectará cuando la bandera (N. del T., «flag») DEB_BUILD_OPTIONS=nostrip está activada y finalizará silenciosamente.

Las líneas 18 a la 26 describen la regla build (y su hija «build-stamp»), que ejecuta make con el propio Makefile de la aplicación para compilar el programa. Si el programa utiliza las utilidades de configuración de GNU para construir los binarios, por favor, asegúrate de leer `/usr/share/doc/autotools-dev/README.Debian.gz`. Hablaremos sobre el ejemplo comentado docbook-to-man más adelante en 'manpage.1.ex, manpage.sgml.es' en la página 28.

La regla «clean» (limpiar, N. del T.), como se especifica en las líneas 28 a la 36, limpia cualquier binario innecesario o cosas generadas automáticamente, dejadas después de la construcción del paquete. Esta regla debe funcionar en todo momento (¡incluso cuando el árbol de fuentes *esté* limpio!), así que, por favor, usa las opciones que fuercen a hacer cosas (por ejemplo para rm, sería «-f»), o ignora los valores devueltos (con un «-» al principio de la orden).

El proceso de instalación, la regla «install», comienza en la línea 38. Básicamente ejecuta la regla «install» del Makefile del programa, pero lo instala en el directorio `$(CURDIR)/debian/gentoo`. Esta es la razón por la que especificamos $(DESTDIR) como el directorio raíz de instalación del Makefile de gentoo.

Como sugiere el comentario, la regla «binary-indep», en la línea 48, se usa para construir paquetes independientes de arquitectura. Como no tenemos ninguno, aquí no se hará nada.

Lo siguiente es la regla «binary-arch», en las líneas 52 a 79, en la que ejecutamos varias pequeñas utilidades del paquete debhelper que nos permiten hacer diversas operaciones en nuestro paquete para que cumpla las normas de Debian.

Si tu paquete es del tipo «Architecture: all» necesitarás incluir todas las órdenes para crear el paquete bajo esta regla, y dejar la siguiente regla («binary-arch») vacía en su lugar.

Los nombres comienzan con dh_ y el resto del nombre es la descripción de lo que la utilidad en particular realmente hace. Es todo más o menos auto-explicativo, pero a continuación tienes algunos añadidos a las explicaciones:

- `dh_testdir(1)` comprueba que estás en el directorio correcto (esto es, el directorio raíz de la distribución de las fuentes),
- `dh_testroot(1)` comprueba que tienes permisos de superusuario que son necesarios para las reglas «binary-arch», «binary-indep» and «clean»,
- `dh_installman(1)` copiará todas las páginas de manual que encuentre en el paquete fuente en el paquete, sólo has de indicarle donde están de forma relativa, desde el nivel más alto del directorio de codigo.
- `dh_strip(1)` elimina las cabeceras de depuración de los ficheros ejecutables para hacerlos más pequeños,
- `dh_compress(1)` comprime las páginas de manual y los ficheros de documentación que sean más grandes de 4 kB con `gzip(1)`,
- `dh_installdeb(1)` copia los ficheros relativos al paquete (es decir, los guiones del desarrollador que mantiene el paquete) bajo el directorio `debian/gentoo/DEBIAN`,
- `dh_shlibdeps(1)` calcula las dependencias de los ejecutables y bibliotecas con las bibliotecas compartidas,
- `dh_gencontrol(1)` genera e instala el fichero de control en `debian/gentoo/DEBIAN`,
- `dh_md5sums(1)` genera las sumas de comprobación MD5 para todos los ficheros del paquete.

Para información más completa de lo que hacen cada uno de estos guiones dh_* , y qué otras opciones tienen, por favor lee sus páginas de manual respectivas. Hay otros guiones con la misma nomenclatura (dh_*) que no se han mencionado aquí, pero pueden serte útiles. Si los necesitas, lee la documentación de debhelper.

La sección binary-arch es en una de las que deberías comentar o eliminar las líneas que llamen a funciones que no necesites. Para gentoo, comentaré de ejemplos, cron, init, man e info, simplemente porque gentoo no las necesita. Tan sólo, en la línea 68, reemplazaré «ChangeLog» con «FIXES», porque este es el nombre del fichero de cambios de las fuentes.

Las últimas dos líneas (junto con otras que no se explican) son cosas más o menos necesarias, sobre las que puedes leer en el manual de make, y las normas. Por ahora no es importante que sepas nada de ellas.

Capítulo 5

Otros ficheros en el directorio debian/.

Verás que existen otros ficheros en el subdirectorio debian/, muchas de los cuales tendrán el sufijo «.ex», que indica que son ejemplos. Echale un vistazo a todos. Si lo deseas o necesitas usar alguna de estas características:

- revisa todo la documentación relacionada (sugerencia: las normas de Debian),
- si es necesario, modifica los ficheros para ajustarlos a tus necesidades,
- renómbralos para eliminar el sufijo «ex.», si lo tiene,
- renómbralos para eliminar el prefijo «.ex», si lo tiene,
- modifica el fichero «rules» si fuera necesario.

Algunos de los ficheros que se usan habitualmente se detallan en las secciones que siguen.

5.1. README.debian (LÉEME.debian, N. del T.)

Cualquier detalle extra o discrepancias entre el programa original y su versión debianizada debería documentarse aquí.

dh_make crea una por omisión, y éste es su aspecto:

```
gentoo for Debian
-----------------

<possible notes regarding this package - if none, delete this file>

 -- Josip Rodin <joy-mg@debian.org>, Wed, 11 Nov 1998 21:02:14 +0100
```

Dado que no tenemos que poner nada aquí - está permitido borrarlo.

5.2. conffiles

Una de las cosas más molestas de los programas es cuando pasas mucho tiempo y esfuerzo adaptando un programa y una actualización destroza todos tus cambios. Debian resuelve este problema marcando los ficheros de configuración de forma que cuando actualizas un paquete se te pregunta si deseas mantener la nueva configuración o no.

Eso se consigue poniendo la ruta completa a cada fichero de configuración (se encuentran generalmente en /etc), una por línea, en un fichero llamado «conffiles» (abreviatura de ficheros de confguración, N. del T.). Gentoo tiene un fichero de configuración, /etc/gentoorc, y meteremos éste en el fichero `conffiles`.

En el caso de que tu programa utilice ficheros de configuración pero también los reescriba él mismo es mejor no marcarlos como «conffiles». Si lo haces, dpkg informará a los usuarios que verifiquen los cambios de estos ficheros cada vez que lo actualicen.

También deberías considerar no marcar el fichero como un conffile si el programa que estás empaquetando requiere que cada usuario modifique su fichero de configuración para poder trabajar.

Puedes tomar ejemplos de ficheros de configuración de los guiones ya existentes de desarrolladores, para más información consulta 'postinst.ex, preinst.ex, postrm.ex y prerm.ex' en la página 30.

Puedes eliminar el fichero `conffiles` del directorio debian/ si tu programa no tiene «conffiles».

5.3. cron.d.ex

Si tu paquete requiere tareas periódicas para funcionar adecuadamente, puedes usar este fichero como patrón.

Ten en cuenta que ésto no incluye la rotación de archivos de registro, para hacer eso consulta `dh_installlogrotate(1)` y `logrotate(8)`.

Elimina el fichero si el paquete no utiliza dichas tareas.

5.4. dirs

Este fichero especifica los directorios que se necesitan pero que por alguna razón no se crean en un proceso de instalación normal («make install»).

Por omisión, tiene este aspecto:

```
1 usr/bin
2 usr/sbin
```

Observa que la barra precedente no está incluida. Normalmente lo cambiaríamos a algo así:

```
1 usr/bin
2 usr/man/man1
```

pero estos directorios ya se crean en el Makefile, así que no necesitaremos este fichero y lo podremos borrar.

5.5. docs

Este fichero especifica los nombres de los ficheros de documentación que dh_installdocs instalará en el directorio temporal.

Por omisión, se incluirán todos los ficheros existentes en los directorios de más alto nivel del código que se llamen «BUGS», «README*», «TODO» etc.

También incluiré algunos otros para gentoo:

```
BUGS
CONFIG-CHANGES
CREDITS
ONEWS
README
README.gtkrc
TODO
```

También podemos eliminar este fichero y en su lugar listar estos ficheros en la línea de órdenes de `dh_installdocs` en el fichero `rules`, de esta forma:

```
dh_installdocs BUGS CONFIG-CHANGES CREDITS ONEWS README \
               README.gtkrc TODO
```

Es posible que no tengas ninguno de estos ficheros en las fuentes de tu paquete. Puedes eliminar este fichero si este es tú caso. Pero no elimines la llamada a `dh_installdocs` desde el fichero `rules` porque también se usa para instalar el fichero `copyright` entre otras cosas.

5.6. emacsen-*.ex

Si tu paquete proporciona ficheros Emacs que pueden ser compilados a bytes en el momento de la instalación, puede usar estos ficheros.

`dh_installemacsen(1)` los instala en el directorio temporal, así que no olvides descomentar esta línea en el fichero `rules` si los usas.

Elimínalos si no los necesitas.

5.7. init.d.ex

Si tu paquete es un demonio que necesita ejecutarse en el arranque del sistema, obviamente has desatendido mi recomendación inicial, ¿o no? :-)

Este fichero es prácticamente un esqueleto genérico para un fichero de guiones en `/etc/init.d/`, así que probablemente tendrás que editarlo y mucho. `dh_installinit(1)` lo instalará en el directorio temporal.

Elimina el fichero si no lo necesitas.

5.8. manpage.1.ex, manpage.sgml.es

El programa debería tener una página de manual. Cada uno de estos ficheros es una plantilla que puedes rellenar en el caso de que no tengas una.

Las páginas de manual se escriben normalmente con `nroff(1)`. El ejemplo `manpage.1.ex` está también escrito con nroff. Consulta la página de manual `man(7)` para una breve descripción de como editar el fichero.

Por otro lado, puede que prefieras escribir usando SGML en lugar de nroff. En este caso, puedes usar la plantilla `manpage.sgml.ex`. Si haces esto, tendrás que:

- instalar el paquete `docbook-to-man`
- añadir `docbook-to-man` a la línea de `Build-Depends` en el fichero de `control`
- eliminar el comentario de la llamada a docbook-to-man en la regla «build» de tu fichero `rules`

¡Y recuerda renombrar el fichero a algo como `gentoo.sgml`!

La página final del nombre debería incluir el nombre del programa que está documentando, asi que lo renombraremos de “manpage” a “gentoo”. El nombre del fichero incluye también “.1” como primer sufijo, lo que significa que es una página de manual para una programa de usuario. Asegurate de verificar que esa sección es la correcta. Aquí tienes una pequeña lista de las secciones de las páginas de manual.

```
Sección |     Descripción        |     Notas
   1     Ordenes de Usuario       Programas o guiones ejecutables.
   2     Llamadas al Sistema      Funciones que ofrece el núcleo.
   3     Llamadas a Bibliotecas   Funciones dadas por las bibliotecas del
   4     Ficheros Especiales      Generalmente se encuentran en /dev.
   5     Formatos de Fichero      Por ejemplo, el formato del /etc/passwd.
   6     Juegos                   U otros programas frívolos.
   7     Paquetes de Macros       Como las macros de man.
   8     Administración del Sist. Programas que sólo suele ejecutar el sup
   9     Rutinas del Núcleo       Llamadas al sistema no estándar.
```

Así que la página de manual de gentoo debería llamarse `gentoo.1`. Para los programas de X puedes añadir una «x» a la sección, por ejemplo, `gentoo.1x`. No había una página de manual gentoo.1 en el paquete fuente asi que la escribí usando la información del ejemplo y de los documentos del programador original.

5.9. menu.ex

Los usuarios de X Windows suelen tener un gestor de ventanas con menús que pueden adaptarse para lanzar programas. Si tienen instalado el paquete `menu` de Debian, se creará un conjunto de menús para cada programa del sistema para ellos.

Éste es el fichero `menu.ex` que dh_make crea por omisión:

```
?package(gentoo):needs="X11|text|vc|wm" section="Apps/lea-manual-menu"\
  title="gentoo" command="/usr/bin/gentoo"
```

El primer campo tras la coma («needs») son las necesidades, y especifica qué tipo de interfaz necesita el programa. Cambia ésta a una de las alternativas que se listan, como por ejemplo «text» o «X11».

Lo siguiente («section») es la sección donde deberían aparecer la entrada del menú y del submenú. La lista actual de secciones está en: `/usr/share/doc/debian-policy/menu-policy.html/ch2.html#s2.1`

El campo «title» es el nombre del programa. Puedes comenzar este en mayúsculas si lo quieres, pero hazlo lo más corto que puedas.

Finalmente, el campo «command» es la orden que ejecuta el programa.

Ahora cambiaremos la entrada del menú por ésta:

```
?package(gentoo): needs="X11" section="Apps/Tools" title="Gentoo" command="
```

También puedes añadir otros campos como son «longtitle» (título largo), «icon» (icono), «hints» (pistas), etc. Para más información consulta `menufile(5)`, `update-menus(1)` y `/usr/share/doc/debian-policy/menu-policy.html/`.

5.10. watch.ex

Este fichero se usa para configurar los programas `uscan(1)` y `uupdate(1)` (en el paquete `devscripts`), que se usan para vigilar el servidor de donde obtuviste las fuentes originales.

Esto es lo que he puesto yo:

```
# watch control file for uscan
# Site Directory Pattern Version Script
ftp.obsession.se /gentoo gentoo-(.*)\.tar\.gz debian uupdate
```

Pista: conéctate a Internet, e intenta ejecutar el programa «uscan» en el directorio donde has creado el fichero. Consulta la página de manual para más detalles.

5.11. ex.package.doc-base

Si tu paquete tiene documentación aparte de las páginas de manual y documentos «info», deberías usar el fichero «`doc-base`» para registrarla, así el usuario puede encontrarlos con `dhelp(1)`, `dwww(1)` o `doccentral(1)`.

Esto incluye generalmente ficheros HTML, PS y PDF que se instalen en `/usr/share/doc/nombre_de_paquete/`.

Así es como el fichero doc-base de gentoo `gentoo.doc-base` debe ser:

```
Document: gentoo
Title: Gentoo Manual
Author: Emil Brink
Abstract: This manual describes what Gentoo is, and how it can be used.
Section: Apps/Tools

Format: HTML
Index: /usr/share/doc/gentoo/html/index.html
Files: /usr/share/doc/gentoo/html/*.html
```

Para información sobre el formato del fichero revisa `install-docs(8)` y el manual de `doc-base` en `/usr/share/doc/doc-base/doc-base.html/index.html`.

Encontrará más detalles de la instalación de documentación adicional en ‘Instalación en un subdirectorio’ en la página 9.

5.12. postinst.ex, preinst.ex, postrm.ex y prerm.ex

Estos ficheros se llaman guiones del desarrollador («maintainer scripts», N. del T.), y son guiones que se colocan en el área de control del paquete y que `dpkg` ejecuta cuando tu paquete se instala, se actualiza o se elimina.

Por ahora, deberías intentar evitar editar manualmente estos guiones si puedes porque suelen hacerse muy complejos. Para más información lee el manual de normas, capítulo 6, y echa un vistazo a los ejemplos dados por dh_make.

Capítulo 6

Construir el paquete

Deberíamos estar preparados para construir el paquete.

6.1. Reconstrucción completa

Entra en el directorio principal del programa y ejecuta la siguiente orden:

```
dpkg-buildpackage -rfakeroot
```

Esto lo hará todo por tí:

- limpia el árbol del código (debian/rules clean), usando `fakeroot`
- construye el paquete de código (dpkg-source -b)
- construye el programa (debian/rules build)
- construye el paquete binario (debian/rules binary), usando `fakeroot`
- firma el fichero fuente `.dsc`, usando `gnupg`
- crea y firma el fichero de subida `.changes`, usando `dpkg-genchanges` y `gnupg`

Lo único que se te pedirá es que escribas tu contraseña secreta de la clave GPG, dos veces.

Después de hacer todo esto, verás las siguientes líneas en el directorio encima del que está (`~/gentoo/`):

- *gentoo_0.9.12.orig.tar.gz*

  Este es el código fuente original comprimido, simplemente se ha renombrado para seguir los estándares de Debian. Nótese que ha sido creado usando la opción «-f» de `dh_make` cuando lo ejecutamos en el inicio.

- *gentoo_0.9.12-1.dsc*

  Este es un sumario de los contenidos del código fuente. Este fichero se genera a partir del fichero de «control» y se usa cuando se descomprimen las fuentes con `dpkg-source(1)`. Este fichero está firmado con GPG de forma que cualquiera pueda estar seguro de que es realmente suyo.

- *gentoo_0.9.12-1.diff.gz*

  Este fichero comprimido contiene todos y cada uno de los cambios que hizo al código fuente original, en un formato conocido como «diff unificado». El programa que lo hace y lo usa es `dpkg-source(1)`. Precaución: si no renombras el archivo comprimido original nombre_de_paquete_versión.orig.tar.gz ¡`dpkg-source` fallará al generar el fichero .diff.gz!

  Si alguien quiere volver a crear tu paquete desde cero, puede hacerlo fácilmente usando los tres ficheros de arriba. El proceso de extracción es trivial: sólo se debe copiar los tres ficheros en algún lado y ejecutar `dpkg-source -x gentoo_0.9.12-1.dsc`.

- *gentoo_0.9.12-1_i386.deb*

  Este es el paquete binario completo. Puedes usar `dpkg` para instalar o eliminar tanto este paquete como cualquier otro.

- *gentoo_0.9.12-1_i386.changes*

  Este fichero describe todos los cambios hechos en la revisión actual del paquete, y se usa por los programas de gestión del archivo FTP para instalar los paquetes binarios y fuentes en él. Se genera parcialmente a partir del fichero «changelog» y el fichero «.dsc». Este fichero está firmado con GPG, de forma que cualquiera puede estar aún más seguro de que es realmente tuyo.

  Mientras sigues trabajando en el paquete, éste cambiará su comportamiento y se le añadirán nuevas funciones. Las personas que descarguen tu paquete pueden leer este fichero y ver qué ha cambiado. Los programas de mantenimiento del archivo de Debian, también enviarán el contenido de este fichero a la lista de correo debian-devel-changes.

Las largas listas de números en los ficheros .dsc y .changes son las sumas MD5 para los ficheros. Las personas que descarguen estos ficheros pueden comprobarlos con `md5sum(1)` y si los números no coinciden, sabrán que el fichero está corrupto o ha sido modificado.

6.2. Reconstrucción rápida

Con un paquete grande, puede que no quieras recompilar desde cero cada vez que tocas un detalle en el fichero `debian/rules`. Para propósitos de prueba, puedes hacer un fichero .deb sin necesidad de recompilar las fuentes originales de esta forma:

```
fakeroot debian/rules binary
```

Una vez que has terminado la puesta a punto, recuerda reconstruir el paquete siguiendo el procedimiento adecuado que está arriba. Puede que no seas capaz de enviar correctamente el paquete si intentas enviar los archivos .deb construidos de esta forma.

6.3. La orden `debuild`

Puedes automatizar aún más el proceso de construcción de paquetes con la orden `debuild`. Veáse `debuild(1)`.

La personalización de la orden debuild puede hacerse a través de `/etc/devscripts.conf` o `~/.devscripts`. Te sugiero al menos los siguientes valores:

```
DEBSIGN_KEYID="Tu_ID_clave_GPG"
DEBUILD_DPKG_BUILDPACKAGE_OPTS="-i -ICVS -I.svn"
```

Con estos valores, puedes construir paquetes siempre con tu clave GPG y evitar incluir componentes no deseados. (Esto también es bueno para patrocinar). Por ejemplo, limpiar el código y reconstruir el paquete desde una cuenta de usuario es tan simple como:

```
debuild clean
debuild
```

6.4. Los sistemas `dpatch` y `quilt`

El uso de las órdenes `dh_make` y `dpkg-buildpackage` creará un gran fichero `diff.gz` que contendrá los archivos de mantenimiento del paquete en `debian/` así como los parches de los ficheros fuente. Este tipo de paquetes es un poco engorroso de inspeccionar y entender para cada una de las modificaciones de código posteriores. Esto no es muy bueno. [1]

Se han propuesto y se utilizan distintos métodos para la gestión de conjuntos de parches en Debian. Los sistemas `dpatch` y `quilt` son los dos más simples de todos los propuestos. Otros son dbs, cdbs, etc.

Un paquete que haya sido empaquetado correctamente con el sistema `dpatch` o `quilt` tiene las modificaciones al código fuente claramente documentadas como un conjunto de ficheros parche de tipo «-p1» en `debian/patches/` y el árbol de código permanece más alládel directorio `debian/`. Si estás pidiendo a un patrocinador que suba tu paquete, este clara separación y documentación de los cambios son muy importantes para acelerar la revisión del paquete por parte del patrocinador. El modo de empleo de `dpatch` y `quilt` se explica en `dpatch(1)`, `dpatch-edit-patch(1)` y `quilt(1)`. Ambos programas ofrecen ficheros que

[1] Si no eres todavía un Desarrollador de Debian y le pides a tu patrocinador que te suba un paquete tras revisarlo, deberías hacer el paquete lo más fácil posible para que él pueda revisarlo.

se pueden incluir en `debian/rules`: `/usr/share/dpatch/dpatch.make` y `/usr/share/quilt/quilt.make`.

Cuando alguien (incluyéndote a ti) proporciona un parche para las fuentes, modificar el paquete con es muy sencillo:

- Edita el parche para crear un parche -p1 sobre el árbol el código fuente.
- En el caso de `dpatch`, añade una cabecera empleando la orden `dpatch patch-template`.
- Pon ese fichero en `debian/patches`
- Añade el nombre de fichero de este parche a `debian/patches/00list` (en `dpatch`) o `debian/patches/series` (en `quilt`).

Además, `dpatch` puede crear parches dependientes de la arquitectura usando macros CPP.

6.5. Incluir `orig.tar.gz` para subir

Cuando subes por primera vez un paquete al archivo, necesitas incluir las fuentes originales `orig.tar.gz`. Si la versión del paquete no es una revisión de Debian `-0` o `-1`, debes proporcionarle la opción «`-sa`» a la orden `dpkg-buildpackage`. Por otro lado, la opción «`-sd`» forzará la exclusión del código original `orig.tar.gz`.

Capítulo 7

Cómo comprobar tu paquete para encontrar fallos

7.1. Los paquetes `lintian` y `linda`

Ejecuta `lintian(1)` y `linda(1)` sobre tu fichero de cambios .changes. Estos programas comprobarán muchos errores comunes al empaquetar. Las órdenes es:

```
lintian -i gentoo_0.9.12-1_i386.changes
linda -i gentoo_0.9.12-1_i386.changes
```

Por supuesto, cambia el nombre de fichero con el nombre del fichero de cambios generado por tu paquete. Si parece que hay algunos errores (líneas que comienzan por E:), lee la explicación (líneas N:), corrige errores, y reconstruye como se describe en 'Reconstrucción completa' en la página 31. Las líneas que comienzan con W: son sólo avisos (warnings, N. del T.), así que afina el paquete o verifica que los avisos son falsos (y haz que lintian los acepte, consulta la documentación para más detalles).

Observa que puedes construir el paquete con `dpkg-buildpackage` y ejecutar `lintian` y `linda` todo con sólo una orden si utilizas `debuild(1)`.

7.2. La orden `mc`

Puedes descomprimir el contenido del paquete `*.deb` con la orden `dpkg-deb(1)`. Puedes listar el contenido de un paquete Debian con `debc(1)`.

Este proceso puede ser muy intuitivo si empleamos un gestor de ficheros como `mc(1)`, que permite visionar tanto el contenido del paquete `*.deb`, como el de los ficheros `*.diff.gz` y `*.tar.gz`.

Vigila que no haya ficheros innecesarios extra o de tamaño cero, tanto en el binario como en el paquete fuente. A veces, hay cosas que no se limpiaron adecuadamente, debes ajustar tu fichero «rules» para arreglar esto.

Pista: «`zgrep ^+++ ../gentoo_0.9.12-1.diff.gz`» te dará una lista de tus cambios o contribuciones a las fuentes, y «`dpkg-deb -c gentoo_0.9.12-1_i386.deb`» o «`debc gentoo_0.9.12-1_i386.changes`» listará los ficheros en el paquete binario.

7.3. La orden `debdiff`

Puedes comparar la lista de ficheros de dos paquetes binarios de Debian con la orden `debdiff(1)`. Este programa es útil para verificar que no hay ficheros que se hayan cambiado de sitio o eliminado por error, y que no se ha realizado ningún otro cambio no deseado al actualizar el paquete. Puedes comprobar un grupo de ficheros `*.deb` simplemente con «`debdiff paquete-viejo.change paquete-nuevo.change`».

7.4. La orden `interdiff`

Puedes comparar dos ficheros `diff.gz` con la orden `interdiff(1)`. Esto es muy útil para verificar que no se han realizado cambios inadvertidos por el mantenedor al actualizar el paquete. Ejecuta «`interdiff -z paquete-viejo.diff.gz paquete-nuevo.diff.gz`».

7.5. La orden `debi`

Instala el paquete para probarlo tú mismo, por ejemplo, usando la orden `debi(1)` como superusuario. Intenta instalarlo y ejecutarlo en otras máquinas distintas de la tuya, y presta atención para detectar errores o avisos tanto en la instalación como en la ejecución del programa.

7.6. El paquete `pbuilder`

El paquete `pbuilder` es muy útil para conseguir un entorno limpio (chroot) donde verificar las dependencias. Esto asegura una construcción limpia desde el código en los programas que realizan la compilación automática de paquetes para diferentes arquitecturas y evita fallos serios del tipo FTBFS (Fallo al construir desde la fuente o «Fail to Build From Source»), que son siempre del tipo RC (fallos criticos o «release critical»). Para más informacon del paquete debian `auto-builder` véase `http://buildd.debian.org/`.

El uso más básico del paquete `pbuilder` es la ejecución directa de la orden `pbuilder` como administrador. Por ejemplo, puedes construir un paquete si escribes las siguientes órdenes en el directorio donde se encuentran los ficheros `.orig.tar.gz`, `.diff.gz` y `.dsc`.

```
root # pbuilder create # si se ejecuta por segunda vez, pbuilder update
root # pbuilder build foo.dsc
```

Los paquetes recién construidos se pueden encontrar en `/var/cache/pbuilder/result/` y el propietario será el usuario administrador.

La orden `pdebuild` te ayuda a usar las funciones del paquete `pbuilder` como usuario sin permisos de administración. Desde el directorio raíz del código fuente, con el archivo `orig.tar.gz` en el directorio padre, escribe las siguientes órdenes:

```
$ sudo pbuilder create # si se ejecuta por segunda vez, sudo pbuilder updat
$ pdebuild
```

Los paquetes construidos se pueden encontrar en `/var/cache/pbuilder/result/` y el propietario no será el administrador. [1]

Si deseas añadir fuentes de apt para que las utilice el paquete `pbuilder`, configura `OTHERMIRROR` en `~/.pbuilderrc` o `/etc/pbuilderrc` y ejecuta (para sarge):

```
$ sudo pbuilder update --distribution sarge --override-config
```

Es necesario el uso de `--override-config` para actualizar las fuentes de apt dentro del entorno chroot.

Véase http://www.netfort.gr.jp/~dancer/software/pbuilder.html, `pdebuild(1)`, `pbuilderrc(5)`, y `pbuilder(8)` .

[1] Actualmente, te sugiero configurar el sistema dando al directorio `/var/cache/pbuilder/result/` permisos de escritura a los usuarios, y editando `~/.pbuilderrc` o `/etc/pbuilderrc` para incluir:

```
AUTO_DEBSIGN=yes
```

Esto te permitirá firmar los paquetes generados con la clave secreta GPG que tienes en `~/.gnupg/`. Puesto que el paquete `pbuilder` está aun en desarrollo, tendrás que comprobar como funciona la configuración consultando la última documentación oficial.

Capítulo 8

Enviar el paquete

Ahora que has probado tu nuevo paquete en profundidad, estarás preparado para comenzar el proceso de nuevo desarrollador de Debian tal y como se describe en: http://www.debian.org/devel/join/newmaint

8.1. Enviar al archivo de Debian

Tendrás subir el paquete al archivo de Debian una vez que llegues a ser un desarrollador oficial. Puedes hacer esto manualmente, pero es más fácil hacerlo con las herramientas automáticas ya disponibles como `dupload(1)` o `dput(1)`. A continuación describiremos como hacerlo con `dupload`.

En primer lugar, tienes que crear un fichero de configuración de dupload. Puedes hacerlo editando el fichero general del sistema `/etc/dupload.conf`, o creando tu propio fichero `~/.dupload.conf` con lo que tu quieras cambiar. Pon algo como esto en el fichero:

```
package config;

$default_host = "anonymous-ftp-master";

$cfg{'anonymous-ftp-master'} = {
      fqdn => "ftp-master.debian.org",
      method => "ftp",
      incoming => "/pub/UploadQueue/",
      # files pass on to dinstall on ftp-master which sends emails itself
      dinstall_runs => 1,
};

1;
```

Por supuesto, cambia el nombre por el tuyo y lee la página de manual `dupload.conf(5)` para comprender qué significa cada una de estas opciones.

La opción $default_host es la más problemática, determina cual de las colas de envios se usará por defecto. “anonymous-ftp-master” es la primaria, pero es posible que quieras usar otra más rápida. Para más información sobre las colas de envío, consulta la Referencia del desarrollador, en concreto la sección «Uploading a package», en `/usr/share/doc/developers-reference/ch-pkgs.en-us.iso-8859-1.html#s-upload`

Ahora conecta con tu proveedor de Internet, y ejecuta la orden:

```
dupload gentoo_0.9.12-1_i386.changes
```

`dupload` comprueba que las sumas md5 coinciden con aquellas en el fichero .changes, y te avisará de rehacer el paquete como se describe en ‘Reconstrucción completa’ en la página 31 para poder enviarlo correctamente.

Si encuentras algún problema con la subida del paquete a `ftp://ftp-master.debian.org/pub/UploadQueue/`, puedes arreglarlo subiendo manualmente a `ftp://ftp-master.debian.org/pub/UploadQueue/` a través de `ftp` un fichero «`*.commands`» firmado con gnupg. [1]. Por ejemplo, usando «`hello.commands`»:

```
-----BEGIN PGP SIGNED MESSAGE-----

Uploader: Roman Hodek <Roman.Hodek@informatik.uni-erlangen.de>
Commands:
 rm hello_1.0-1_i386.deb
 mv hello_1.0-1.dsx hello_1.0-1.dsc

-----BEGIN PGP SIGNATURE-----
Version: 2.6.3ia

iQCVAwUBNFiQSXVhJ0HiWnvJAQG58AP+IDJVeSWmDvzMUphScg1EK0mvChgnuD7h
BRiVQubXkB2DphLJW5UUSRnjw1iuFcYwH/lFpNpl7XP95LkLX3iFza9qItw4k2/q
tvylZkmIA9jxCyv/YB6zZCbHmbvUnL473eLRoxlnYZd3JFaCZMJ86B0Ph4GFNPAf
Z4jxNrgh7Bc=
=pH94
-----END PGP SIGNATURE-----
```

8.2. Enviar a un archivo privado

Puedes crear un repositorio personal de paquetes en `URL=“http://people.debian.org/~nombre_de_c` (si eres desarrollador oficial) con una simple llamada a `dupload -t` *`nombre_de_objetivo`*. Para hacerlo deberías añadir lo siguiente al fichero «`/etc/dupload.conf`»:

[1] Véase `ftp://ftp-master.debian.org/pub/UploadQueue/README`. Como alternativa, puedes usar la orden `dcut` de el paquete `dput`.

```
# Cuenta de desarrollador
$cfg{'nombre_de_objetivo'} = {
        fqdn => "people.debian.org",
        method => "scpb",
        incoming => "/home/nombre_de_cuenta/public_html/package/",
        # No necesitas anunciarlo
        dinstall_runs => 1,
};
$cfg{'nombre_de_objetivo'}{preupload}{'changes'} = "
        echo 'mkdir -p public_html/package' | ssh people.debian.org  2>/dev/n
        echo '¡Directorio de paquetes creado!'";

$cfg{'nombre_de_objetivo'}{postupload}{'changes'} = "
        echo 'cd public_html/package ;
        dpkg-scanpackages . /dev/null >Packages || true ;
        dpkg-scansources . /dev/null >Sources || true ;
        gzip -c Packages >Packages.gz ;
        gzip -c Sources >Sources.gz ' | ssh people.debian.org  2>/dev/null ;
        echo '¡Archivo de paquetes creado!'";
```

Aquí, el repositorio APT se construye mediante una simple ejecución remota con SSH. Los ficheros de sobreescritura que necesitan `dpkg-scanpackages` y `dpkg-scansources` se especifican como `/dev/null`. Esta técnica la puede emplear alguien que no es desarrollador de Debian para almacenar sus paquetes en su página personal. También se pueden usar `apt-ftparchive` o otros programas para crear un repositorio APT.

Capítulo 9

Actualizar el paquete

9.1. Nueva revisión Debian del paquete

Supongamos que se ha creado un informe de fallo en tu paquete con el número #54321, y que describe un problema que puedes solucionar. Para crear una nueva revisión del paquete, necesitas:

- Corregir, por supuesto, el problema en las fuentes del paquete.
- Añadir una nueva revisión en el fichero de cambios (changelog, N. del T.) de Debian, con «`dch -i`», o explícitamente con «`dch -v <versión>-<revisión>`» y entonces insertar los comentarios con tu editor favorito.

  Sugerencia: ¿Como obtener la fecha fácilmente en el formato requerido? Usa «`822-date`», o «`date -R`».
- Incluir una breve descripción del error y su solución en la entrada del fichero de cambios, seguido por: «Closes: #54321». De esta forma, el informe del error será automágicamente cerrado por los programas de gestión del archivo en el momento en que tu paquete se acepte en el archivo de Debian.
- Repite lo que hiciste en 'Reconstrucción completa' en la página 31, 'Cómo comprobar tu paquete para encontrar fallos' en la página 35, y 'Enviar el paquete' en la página 39. La diferencia es que esta vez, las fuentes originales del archivo no se incluirán, dado que no han cambiado y ya existen en el archivo de Debian.

9.2. Nueva versión del programa fuente (básico)

Ahora consideremos una situación diferente y algo más complicada: ha salido una versión nueva de las fuentes originales, y, por supuesto, deseas empaquetarla. Debes hacer lo siguiente:

- Descarga las nuevas fuentes y pon el archivo tar (pongamos que se llama `gentoo-0.9.13.tar.gz`) un directorio por encima del antiguo árbol de fuentes (por ejemplo ~/gentoo/).
- Entra en el antiguo directorio de las fuentes y ejecuta:

```
uupdate -u gentoo-0.9.13.tar.gz
```

  Por supuesto, reemplaza este nombre de fichero con el nombre de las fuentes de tu programa. `uupdate(1)` renombrará apropiadamente este fichero tar, intentará aplicar los cambios de tu fichero `.diff.gz` previo y actualizará el nuevo fichero `debian/changelog`.
- Cambia al directorio «`../gentoo-0.9.13`», el nuevo directorio fuente del paquete, y repite la operación que hiciste en 'Reconstrucción completa' en la página 31, 'Cómo comprobar tu paquete para encontrar fallos' en la página 35, y 'Enviar el paquete' en la página 39.

Observa que si has puesto el fichero «`debian/watch`» como se describe en 'watch.ex' en la página 29, puedes ejecutar `uscan(1)` para buscar automáticamente fuentes revisadas, descargarlas, y ejecutar `uupdate`

9.3. Nueva versión de las fuentes (realista)

Cuando prepares paquetes para el archivo de Debian, debes comprobar los paquetes resultantes en detalle. A continuación, tienes un ejemplo más realista de este procedimiento.

1 Verificar los cambios en las fuentes.
   - De las fuentes, lee los ficheros `changelog`, `NEWS`, y cualquier otra documentación que se haya publicado con la nueva versión.
   - Ejecuta «`diff -urN`» entre las fuentes viejas y las nuevas para obtener una visión del alcance de los cambios, donde se ha trabajado más activamente (y por tanto donde podrían aparecer nuevas erratas), y también busca cualquier cosa que pudiera parecer sospechosa.

2 Porta el paquete Debian viejo a la nueva versión.
   - Descomprime el código fuente original y renombra la raíz del árbol de las fuentes como `<nombrepaquete>-<versión_original>/` y haz «`cd`» en este directorio.
   - Copia el código fuente en el directorio padre y renombrarlo como `<nombrepaquete>_<versión_original>.orig.tar.gz` .
   - Aplica el mismo tipo de modificación a el nuevo código que al viejo. Algunos posibles métodos son:
     - orden «`zcat /path/to/<nombrepaquete>_<versión-vieja>.diff.gz | patch -p1`»,
     - orden «`uupdate`»,
     - orden «`svn merge`» si gestionas el código con un repositorio Subversion o,

    - simplemente copia el directorio `debian/` del árbol de código viejo si se empaquetó con `dpatch` o `quilt`.
  - Conserva las entradas viejas del fichero «changelog» (puede parecer obvio, pero se han dado casos...)
  - La nueva versión del paquete es la versión original añadiéndole el número de revisión de Debian `-1`, por ejemplo, '`0.9.13-1`'.
  - Añade una entrada en el fichero «changelog» para esta nueva versión al comienzo `debian/changelog` que ponga «New upstream release» (nueva versión original, N. del T.). Por ejemplo, «`dch -v 0.9.13-1`».
  - Describe de forma resumida los cambios *en* la nueva versión del código fuente que arreglan fallos de los que ya se ha informado y cierra esos fallos en el fichero «changelog».
  - Describe de forma resumida los cambios hechos *a* la nueva versión del código por el mantenedor para arreglar fallos de los que se ha informado y cierra esos fallos en el fichero «changelog».
  - Si el parche/fusión no se aplicó limpiamente, inspecciona la situación para determinar qué ha fallado (la clave está en los ficheros `.rej`). A menudo el problema es que un parche que has aplicado a las fuentes se ha integrado en el código fuente original, y, por lo tanto, el parche ya no es necesario.
  - Las actualizacioens de versión deberían ser silenciosas y no intrusivas (los usuarios sólo deberían advertir la actualización al descubrir que se han arreglado viejos fallos y porque se han introducido algunas nuevas características). [1]
  - Si necesitas añadir plantillas eliminadas por alguna razón, puedes ejecutar `dh_make` otra vez en el mismo directorio ya «debianizado», con la opción `-o`. Una vez hecho esto edítalo como sea necesario.
  - Deberías reconsiderar todos los cambios introducidos para Debian: elimina aquello que el autor original haya incorporado (de una forma u otra) y recuerda mantener aquellos que no hayan sido incorporados, a menos que haya una razón convincente para no incluirlos.
  - Si se ha realizado algún cambio en el sistema de construcción (esperemos que lo supieras desde el primer paso), actualiza los ficheros `debian/rules` y las dependencias de construcción en `debian/control` si es necesario.
3. Construye el nuevo paquete como se describe en 'La orden `debuild`' en la página 33 o 'El paquete `pbuilder`' en la página 36. Es conveniente el uso de `pbuilder`.
4. Comprueba que los paquetes nuevos se han construido correctamente.
  - Ejecuta 'Cómo comprobar tu paquete para encontrar fallos' en la página 35.
  - Ejecuta 'Verificar actualizaciones del paquete' en la página 47.
  - Comprueba de nuevo si se ha arreglado alguno de los fallos que actualmente están abiertos en el sistema de seguimiento de fallos de Debian (BTS) (`http://www.`

[1] Asegúrate de que tu paquete actualiza adecuadamente los ficheros de configuración con `postinst` etc, bien diseñados, para que ¡**no** se hagan cosas que el usuario no quiere! Estas son las mejoras que explican **por qué** la gente escoge Debian. Cuando la actualización es necesariamente intrusiva (por ejemplo, en la nueva versión se han repartido los ficheros de configuración en distintos directorios personales con estructuras totalmente diferentes), puedes considerar hacer que el paquete tome un valor seguro por omisión (por ejemplo, deshabilitando el servicio) y proporcionar la documentación necesaria (tal y como lo exigen las normas: en ficheros `README.Debian` y `NEWS.Debian`) como último recurso. Pero no te molestes en incluir una nota con debconf.

debian.org/Bugs/).
- Comprueba el contenido del fichero «.changes» para asegurarte de que lo estás enviando a la distribución correcta, que se cierran los fallos adecuadamente en el campo «Closes:», que los campos «Maintainer:» y «Changed-By:» coinciden, que el fichero se ha firmado con GPG, etc.

5 Si realizaste algún cambio en el empaquetado durante el proceso, vuelve al segundo paso hasta que todo esté correcto.
6 Si tu envío necesita que se patrocine, asegúrate de comentar cualquier opción especial que se requiera en la construcción del paquete (como «`dpkg-buildpackage -sa -v ...`») y de informar a tu patrocinador, así podrá construirlo correctamente.
7 Si lo envías tú, ejecuta 'Enviar el paquete' en la página 39.

9.4. El archivo `orig.tar.gz`

Si intentas construir los paquetes sólo desde el nuevo código fuente con el directorio `debian/`, sin que exista el fichero `orig.tar.gz` en el directorio padre, acabarás creando un paquete de fuentes nativo sin querer. Estos paquetes se distribuyen sin el fichero `diff.gz`. Este tipo de empaquetamiento sólo debe hacerse para aquellos paquetes que son específicos de Debian, es decir, aquellos que no serían útiles en otra distribución. [2]

Para obtener un paquete no nativo de fuentes que consista tanto en un archivo `orig.tar.gz` como en un archivo `diff.gz`, debes copiar manualmente el archivo tar del código fuente original al directorio padre con el nombre cambiado a `<nombrepaquete>_<versión>.orig.tar.gz`. Igual que como lo hizo la orden `dh_make` en '«Debianización» inicial' en la página 8.

9.5. La orden `cvs-buildpackage` y similares

Deberías considerar el utilizar algún sistema de administración de código para gestióna el proceso de empaquetado. Hay varios guiones adaptados para que puedan utilizarse en algunos de los sistemas de control de versiones más populares.

- CVS
  - `cvs-buildpackage`
- Subversion
  - `svn-buildpackage`

Estas órdenes también automatizan el empaquetado de nuevas versiones del código fuente.

[2] Algunas personas argumentan que, incluso en el caso de paquetes específicos de Debian, es una práctica mejor de empaquetamiento que los contenidos del directorio `debian/` estén en el archivo `diff.gz` en lugar de incluirse en el archivo `orig.tar.gz`.

9.6. Verificar actualizaciones del paquete

Cuando construyas una nueva versión del paquete, deberías hacer lo siguiente para verificar que el paquete puede actualizarse de forma segura:

- actualiza el paquete a partir de la versión previa,
- vuelve a la versión anterior y elimínala,
- instala el paquete nuevo,
- elimínalo y reinstálalo de nuevo,
- púrgalo.

Si el paquete hace uso de unos guiones pre/post/inst/rm complicados, asegúrate de probar éstos con las distintas rutas posibles en la actualización del paquete.

Ten en cuenta que si tu paquete ha estado previamente en Debian, lo más frecuente es que gente actualice el paquete desde la versión que estaba en la última versión de Debian. Recuerda que debes probar también las actualizaciones desde esa versión.

Capítulo 10

Dónde pedir ayuda

Antes de que te decidas a preguntar en lugares públicos, por favor, simplemente RTFM («Lee el dichoso manual», N. del T.), que incluye la documentación en `/usr/share/doc/dpkg`, `/usr/share/doc/debian`, `/usr/share/doc/autotools-dev/README.Debian.gz`, `/usr/share/doc/package/*` y las páginas de man/info para todos los programas mencionados en este documento. Consulta toda la información en `http://nm.debian.org/` y `http://people.debian.org/~mpalmer/debian-mentors_FAQ.html`.

Si tienes dudas sobre empaquetado a las que no has podido encontrar respuesta en la documentación, puedes preguntar en la lista de correo de Debian Mentors `<debian-mentors@lists.debian.org>`. Los desarrolladores más experimentados de Debian, te ayudarán gustosamente, pero ¡leete la documentación antes de preguntar!

Consulta `http://lists.debian.org/debian-mentors/` para más información sobre esta lista de correo.

Cuando recibas un aviso de fallo (sí, avisos de fallos, ¡de verdad!) sabrás que es el momento de indagar en el Sistema de seguimiento de fallos de Debian (`http://www.debian.org/Bugs/`) y leer la documentación de allí para poder tratar los informes de forma eficiente. Te recomiendo la lectura de la Refencia del Desarrollador, en particular el capítulo de «Manejo de Bugs» (Handling Bugs, N. del T.), en `/usr/share/doc/developers-reference/ch-pkgs.en-us.iso-8859-1.html#s-bug-handling`.

Si aún tienes preguntas, házlas en la lista de distribución de Desarrolladores de Debian en `<debian-devel@lists.debian.org>`. Véase `http://lists.debian.org/debian-devel/` para más información sobre esta lista de correo.

Aunque todo funcione bien, es el momento de empezar a rezar. ¿Por qué? Por que en sólo unas horas (o días) usuarios de todo el mundo empezarán a usar tu paquete, y si cometiste algún error crítico centenares de usuarios furiosos de Debian te bombardearán con correos… sólo bromeaba :-)

Relájate y prepárate para recibir informes de fallos, porque hay mucho más trabajo que hacer antes de seguir completamente las Normas de Debian (una vez más lee la *documentación real* para más detalles). ¡Buena suerte!

Apéndice A

Ejemplos.

En este ejemplo mpaquetaremos el código fuente original *gentoo-1.0.2*.tar.gz y subiremos todos los paquetes al *nm_objetivo*.

A.1. Ejemplo de empaquetado sencillo

```
$ mkdir -p /ruta/a # nuevo directorio vacío
$ cd /ruta/a
$ tar -xvzf /ruta/desde/gentoo-1.0.2.tar.gz # obtén la fuente
$ cd gentoo-1.0.2
$ dh_make -e nombre@dominio.com -f /ruta/desde/gentoo-1.0.2.tar.gz
... Responde a las preguntas
... Arregla el árbol de las fuentes
... Si es un paquete que contiene programas guiones, indica en debian/contro
... No borres ../gentoo_1.0.2.orig.tar.gz
$ debuild
... Asegúrate de que no hay ningún aviso
$ cd ..
$ dupload -t nm_objetivo gentoo_1.0.2-1_i386.changes
```

A.2. Ejemplo de empaquetado con `dpatch` y `pbuilder`

```
$ mkdir -p /ruta/a # nuevo directorio vacío
$ cd /ruta/a
$ tar -xvzf /ruta/desde/gentoo-1.0.2.tar.gz
$ cp -a  gentoo-1.0.2 gentoo-1.0.2-orig
$ cd gentoo-1.0.2
$ dh_make -e nombre@dominio.com -f /ruta/de/gentoo-1.0.2.tar.gz
... Responde a las preguntas
```

En un principio, `debian/rules` es así:

```
configure: configure-stamp
configure-stamp:
        dh_testdir
        # Add here commands to configure the package.
        touch configure-stamp
build: build-stamp
build-stamp: configure-stamp
        dh_testdir
        # Add here commands to compile the package.
        $(MAKE)
        #docbook-to-man debian/gentoo.sgml > gentoo.1
        touch $@
clean:
        dh_testdir
        dh_testroot
        rm -f build-stamp configure-stamp
        # Add here commands to clean up after the build process.
        -$(MAKE) clean
        dh_clean
```

Cambia lo siguiente con un editor en `debian/rules` para usar `dpatch` y añade `dpatch` a la línea `Build-Depends:` en el fichero `debian/control`:

```
configure: configure-stamp
configure-stamp: patch
        dh_testdir
        # Add here commands to configure the package.
        touch configure-stamp
build: build-stamp
build-stamp: configure-stamp
        dh_testdir
        # Add here commands to compile the package.
        $(MAKE)
        #docbook-to-man debian/gentoo.sgml > gentoo.1
        touch $@
clean: clean-patched unpatch
        dh_testdir
        dh_testroot
        rm -f build-stamp configure-stamp
        # Add here commands to clean up after the build process.
        -$(MAKE) clean
        dh_clean
patch: patch-stamp
```

```
patch-stamp:
     dpatch apply-all
     dpatch call-all -a=pkg-info >patch-stamp
unpatch:
     dpatch deapply-all
     rm -rf patch-stamp debian/patched
```

Ahora está todo preparado para reempaquetar el árbol de código con el sistema `dpatch` y con la ayuda de `dpatch-edit-patch`.

```
 $ dpatch-edit-patch patch 10_firstpatch
... Arregla el arbol de fuentes con el editor
$ exit 0
... Intenta construir el paquete con «debuild -us -uc»
... Limpia las fuentes con «debuild clean»
... Repite con dpatch-edit-patch hasta que las fuentes compilen.
$ sudo pbuilder update
$ pdebuild
$ cd /var/cache/pbuilder/result/
$ dupload -t nm_objetivo gentoo_1.0.2-1_i386.changes
```