Resumen

El paquete es el componente fundamental de una distribución, pero dentro de éstos hay mucho más de lo que uno pudiera imaginar. En este artículo se analiza la distribución Debian GNU/Linux desde esta perspectiva.

1 Introducción

Es importante conocer la estructura de paquetes que las distribuciones usan, porque sólo así uno es capaz de arreglar los problemas que puedan surgir en su uso diario (corrupción de archivos, instalación de programas fuera de la distribución etc...).

A pesar de que las distribuciones vienen con un buen número de software (Debian GNU/Linux, por ejemplo, cuenta con más de 4000 paquetes de software) a veces interesará instalar software que no es parte de la distribución que aún no ha sido incorporado a ésta; esto incluye paquetes de software que se encuentran en la red y software comercial, e, incluso, aplicaciones creadas por el propio usuario; para estas cosas serán necesario hacerse sus propios paquetes si no se quiere entrar en conflicto con el sistema de paquetes. Por supuesto, si se desea contribuir a las distintas distribuciones con software, evidentemente, se deberá dar los programas convenientemente empaquetados.

También puede ser útil para recompilar paquetes nuevos con librerías antiguas, es el caso, por ejemplo, con los paquetes hamm de Debian 2.0 compilados para libc6, que podrían instalarse en un sistema bo (Debian 1.3.1), que utiliza libc5, sin más que recompilar el código fuente e instalar el paquete (si fuera sólo ésta la única dependencia problemática, el significado de las dependencias se verá más tarde).

En éste artículo se verá en detalle el sistema de paquetes de Debian GNU/Linux desde el punto de vista del formato de paquete (.deb) con objeto de preparar al lector interesado para lo arriba indicado. Hay que tener en cuenta, sin embargo, que el sistema de paquetes es mucho más amplio que sólo un formato de archivo ya que lleva detrás toda una filosofía de "cómo hacer las cosas", que en el caso de Debian es una Política bien definida.

La razón de detallar el sistema de paquetes de Debian es múltiple: por un lado el sistema de paquetes de Debian GNU/Linux es muy versátil, con algunas características que dan uniformidad a la distribución en cuanto a la localización de programas y documentación; asimismo, el sistema Debian GNU/Linux es el más abierto con respecto a la incorporación de desarrolladores (en inglés, 'maintainers') a éste, a diferencia de otras distribuciones comerciales en las cuales la contribución está más limitada. Y finalmente, porque de entre muchos otros formatos de paquetes Debian ofrece más que la mayoría, como puede leer en este informe[1].

2 Sistema de paquetes frente a formato de paquetes

Es necesario diferenciar, en primer lugar, entre lo que es el sistema de paquetes y el formato de paquetes, para no dar lugar a confusión. El sistema de paquetes es el conjunto de reglas propias de una distribución que indican dónde se localizan los programas, cómo se instalan demonios en el sistema, qué ficheros de configuración genéricos hay accesibles por los programas, así como las distintas interacciones entre los paquetes, indicando, por ejemplo, si dos programas tienen incompatibilidades y no pueden coexistir en el mismo sistema (conflictos) o si antes de instalar un programa es necesario tener otro instalado (dependencias).

El formato de los paquetes, por ejemplo los ficheros `.deb` en el caso de Debian o `.rpm` en el caso de RedHat, se suele identificar con el sistema de paquetes. Pero, si bien el sistema condiciona cómo deberán crearse y distribuirse los paquetes (qué reglas han de seguir para instalarse), es posible instalar paquetes de otras distribuciones en nuestro sistema, e incluso podemos encontrar herramientas para hacerlo. Por ejemplo, `alien` es un programa (disponible como paquete en Debian) que, una vez instalado, permite introducir paquetes que no pertenecen a la distribución de Debian (por ejemplo, rpms) ya que "conoce"los distintos formatos y es capaz de "traducirlosª nuestra distribución. El formato, aunque relacionado con el sistema, no es mucho más que eso. Las diferencias entre un `.deb` y un `.rpm` son en esencia similares a las que existen entre un `.zip` y un `.arj`.

Sin embargo, existe un riesgo cuando se mezclan paquetes de distintas distribuciones, y es que su política, esto es, el sistema de paquetes, será distinta. Para poner un ejemplo: Debian y RedHat siguen una política distinta en cuanto a la localización de los programas que ejecutan los demonios en el arranque, aunque ambos siguen el modelo de System V (Slackware sigue el modelo de BSD, colocándolos en otro sitio), RedHat coloca los demonios en el directorio `/etc/rc.d/init.d` y con los enlaces en `/etc/rc.d/rcX.d`, mientras que Debian

[1] `http://kitenet.net/~joey/pkgcomp.html`

lo hace en el `/etc/init.d` con los enlaces en `/etc/rcX.d`; y ésta no es la única diferencia. Es evidente que si instalamos un paquete de una distribución, que proporcione demonios que han de ejecutarse en el arranque en otra, posiblemente no funcionará.

Se lleva un tiempo debatiendo sobre una posible estandarización de los sistemas de paquetes, que quizás se consiga como ya se consiguió homogeneizar la estructura de directorios a través del Linux Filesystem Structure (FSSTND) con lo que es posible que en un futuro habrá mayor compatibilidad entre las distribuciones.

3 Construcción de nuestro primer paquete

Se van a ver, a continuación, los pasos y herramientas necesarios para la creación de un paquete bajo un sistema Debian GNU/Linux. Se va a escoger el paquete hello pues es el que ofrece Debian para mostrar el sistema de construcción de paquetes, que no tiene más que la versión GNU de hello, habitual para los programadores, que se trata un simple programa que escribe "Hello world"por la salida estándar.

Se obtienen primero los tres ficheros fuentes del paquete Debian hello[2], es decir: `hello_x.x.orig.tar.gz`, `hello_x.x-xx.diff.gz` y `hello_x.x-xx.dsc` (donde las 'x' dependerán del número de versión). Todos se pueden encontrar en una distribución de Debian en stable/main/source/misc/[3].

El primero de ellos es el código fuente original, el segundo un fichero con las diferencias entre el árbol fuente original (el directorio donde se encuentra el código fuente) y el árbol fuente Debian, y el tercero es una breve descripción del paquete, que, como se verá después, está firmada con PGP (Pretty Good Privacy, ver más abajo) por la persona que lo ha empaquetado y tiene un valor de control (función hash MD5) de los dos ficheros anteriores para poder detectar si han sido modificados por alguien ajeno al desarrollador (útil para detectar paquetes 'troyanos').

En primer lugar se ejecutará, con los tres ficheros en un mismo directorio, `dpkg-source -x hello_x.x-xxx.dsc`, que realizará un `untar` del fichero original (generando la estructura de directorios del árbol fuente original) y, posteriormente, aplicará el programa `patch` para incorporar las modificaciones que se han hecho en Debian del paquete. Dentro del directorio generado, que será de la forma `nombre_de_paquete-version`, se ejecutará `dpkg-buildpackage`, que, si todo sale bien (como se puede ver aquí), dejará en el directorio anterior, un fichero `hello_xxx.deb` que será el paquete preparado para instalar.

El proceso de construcción del paquete lo realiza la orden `dpkg-buildpackage` y para ello ejecuta, por orden: `dpkg-source`, `debian/rules` (con los métodos clean, build y binary), `dpkg-shlibdeps`, `dpkg-gencontrol`, `dpkg-genchanges`, y PGP, más adelante se verá su significado aunque se pueden ver los distintos pasos in figure **??**.

Es necesario hacer todo esto como root, ya que una serie de las operaciones que se ejecutan necesitan tener los privilegios de este usuario, como es el cambio de propietario de los ficheros (pasan a ser del usuario root y el grupo root generalmente). Esto puede ser un problema cuando un usuario quiera generar un paquete en un sistema en el que carece de estos privilegios. Para esto existe el programa `fakeroot` que hace creer al sistema que el usuario es root, esto no supone ningún problema de seguridad porque en realidad es sólo un engaño para el usuario y sus aplicaciones que, en cualquier caso, no adquieren ninguno de los privilegios del superusuario.

Debian usa PGP (aunque cambiará pronto a GPG) para certificar la autenticidad e integridad de los paquetes, ya que el sistema de inserción de paquetes hechos por desarrolladores de Debian es semi-automático (via varios servidores de ftp anónimo y las máquinas de Debian) y, también es posible que personas ajenas a Debian (u otros desarrolladores de Debian) manden cambios, para, por ejemplo, arreglar errores críticos. Es por tanto importante que los paquetes vayan firmados por el que hizo las modificaciones (`dpkg-buildpackage` llama a PGP al final) y proteger contra modificaciones del paquete que se intenten hacer una vez el maintainer ha dado su versión. Se firma así el fichero `.dsc` que contiene una descripción del paquete y una "huella" de los ficheros anteriormente vistos, esta firma se realiza con una función "hash" muy conocida: MD5; también se firma, si existiera, el fichero `.changes` que contiene las modificaciones realizadas entre una versión.

4 Las herramientas dpkg-xxxxx

Como se ha visto en el ejemplo anterior Debian posee una serie de herramientas que es necesario llamar para construir un paquete. Serán éstas:

- `dpkg-source` empaqueta y desempaqueta los archivos fuentes de un paquete Debian.

[2] `http://packages.debian.org/hello`
[3] `ftp://ftp.es.debian.org/debian/stable/main/source/misc`

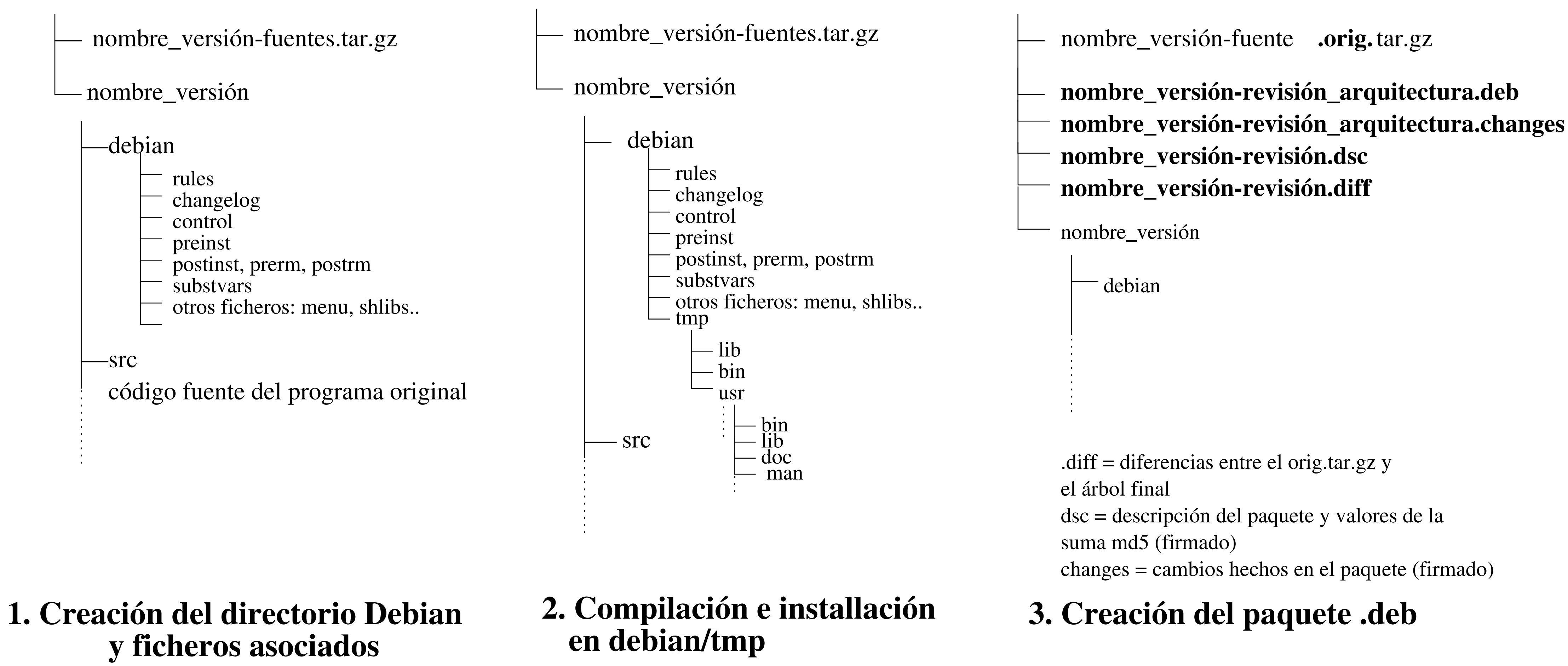

Figura 1: Pasos tomados para construir un paquete

- `dpkg-gencontrol` lee la información de un árbol fuente Debian desempaquetado y genera un paquete binario de control, generando una entrada para éste en el fichero `debian/files`.

- `dpkg-shlibdeps` calcula las dependencias de ejecutables respecto a librerías.

- `dpkg-genchanges` lee la información de un árbol fuente Debian desempaquetado y ya construido, generando un fichero de control de los últimos cambios (un .changes).

- `dpkg-buildpackage` es un script de control que se puede utilizar para automatizar la construcción del paquete.

- `dpkg-distaddfile` añade una entrada de un fichero a `debian/files`.

- `dpkg-parsechangelog` lee el fichero de cambios (`changelog`) de un árbol fuente Debian desempaquetado y genera una salida con la información de estos cambios, convenientemente preparada.

5 El directorio debian/

Debian controla las características y evolución del paquete a través de una serie de ficheros. Cosas tales como la descripción del paquete, las dependencias con otros paquetes, con librerías instaladas, cambios producidos en el paquete, reglas para construir y compilar los binarios del paquete, etc..

Esto se consigue con el directorio `debian/`, que, en principio, es lo único que añade Debian al código fuente original de un paquete. En este directorio se encuentran un conjunto de ficheros (vea la figura **??**) que deben seguir unas reglas definidas en la Política de Paquetes de Debian[4], en la que se explica tanto el contenido de éstos como su formato.

En el fichero control se definen las características del paquete, y es, básicamente, lo que se observa cuando se ejecuta `dpkg -status` sobre un paquete ya instalado o `dpkg -info` sobre uno no instalado (sobre el fichero `.deb`). Sus campos son:

- *Source:* nombre del paquete fuente original.

- *Section:* sección de Debian a la que pertenece (por ejemplo: devel, web, admin...).

- *Priority:* prioridad que tiene este paquete dentro de la distribución (required, important, optional, standard, extra).

- *Maintainer:* nombre y dirección de e-mail del que mantiene el paquete.

- *Standards-Version:* estándar de Debian bajo el cual se ha creado el paquete.

- *Package:* nombre del paquete en Debian.

- *Architecture:* arquitectura para la que se ha creado (i386, alpha, arm, m68k, powerpc, sparc..).

- *Depends:* dependencias con otros paquetes, se indica tanto el paquete como su versión.

- *Conflicts:* paquetes con los que entra en conflicto y que no pueden estar instalados cuando se instala éste.

- *Suggests:* paquetes que mejoran el paquete que se está instalando y que, aunque no son necesarios para su funcionamiento, se recomienda su instalación.

- *Description:* descripción breve (una línea) y larga del contenido del paquete.

El fichero `rules` contiene las reglas para construir el paquete y será la que llamen los programas de construcción de paquetes. Se trata de un 'Makefile', un fichero habitual para aquellos acostumbrados a compilar programas en entornos UNIX. Dentro de este fichero encontramos una serie de reglas y objetivos a cumplir. Dentro de estos últimos podemos destacar varios de importancia:

- *clean*: limpia el árbol de binarios y ficheros temporales. Se ejecutará siempre para asegurse que la compilación/construcción del paquete se hace sobre una base "limpia"

[4] `http://www.es.debian.org/doc/debian-policy`

- *build*: compila las fuentes del programa para obtener los binarios (también otras cosas generadas automáticamente como, a veces, la documentación).

- *binary*: llama a dos subobjetivos: *binary-indep* y *binary-arch*, que van a realizar la instalación del paquete bajo `debian/tmp`, moviendo allí programas, documentación y librerías, cambiando los permisos y propietarios según corresponda. El primero realizará las tareas independientes de arquitectura y el segundo las tareas para una arquitectura determinada.

- *get-orig-source*: especifica una forma de obtener el código fuente original que se ha utilizado para construir el paquete (via ftp).

El fichero `changelog` documenta los cambios hechos en la debianización del programa, estos cambios se refieren a los particulares de Debian no a los que se hagan en el código fuente; en el raíz generalmente habrá un fichero llamado `changelog` que documentará los cambios del programa. Sigue un formato específico, aunque se puede utilizar `dch` o `debchange` (ver más abajo) para modificarlo. Hay que recordar que, generalmente, el que mantiene el paquete (y lo construye) y el autor del programa serán distintos. Aunque Debian tiene paquetes hechos expresamente para este sistema y elaborados por sus desarrolladores, esto no es la norma general, el compromiso principal de Debian[5] es el de hacer disponible programas de libre distribución en un sistema completo y homogéneo.

En `conffiles` se listan los ficheros de configuración que instala el paquete. Esto es necesario para que Debian no sobreescriba ficheros de configuración que el usuario ya ha modificado. En el momento de instalar un programa, si hubiera ficheros de configuración, Debian indicará que son distintos y dará la oportunidad de instalar el nuevo o dejar el anterior, arreglando el problema de que la instalación de una nueva versión del paquete destruya el trabajo realizado en configurarlo.

Los scripts `preinst`, `postinst`, `prerm` y `postrm` son scripts ejecutados por el instalador de paquetes en diversos momentos de su instalación, respectivamente antes (pre) y después (post) de ser instalado (`dpkg -install`) o eliminado (`dpkg -remove`) del sistema. Estos scripts permiten que, en el momento de instalar el paquete, se actualizen ficheros o se configuren los programas.

Finalmente, el fichero `README.debian` contiene detalles o discrepancias entre el paquete original y la versión de Debian. Este fichero se encontrará, una vez instalado en paquete en `/usr/doc/nombre_paquete`, junto a toda la documentación, el copyright y el fichero de cambios (de la versión original y la de Debian).

Existen otros ficheros: `menu`, `init.d`, `crontab`... que pueden usarse para integrar el paquete aún más en el sistema.

6 Algunas herramientas útiles

Existen algunas herramientas que no forman parte de las "estándar"de Debian, pero que pueden resultar útiles a la hora de crear paquetes, dado que simplifican algunas de las tareas comunes a las que nos podemos enfrentar en el momento de hacer un paquete.

Una de éstas es debmake[6], aunque ahora en desuso y poco recomendado, contiene un buen número de herramientas para la creación de paquetes. Por ejemplo, ejecutando `debmake` en el raíz del árbol fuente original, se generará el directorio debian y todos los ficheros de éste, preparados para que el usuario los modifique convenientemente.

Muchas de las utilidades de debmake han sido retiradas de éste ya que, muy posiblemente, deje pronto de existir, estas utilidades se han incorporado, junto con otras, al paquete `devscripts` (`http://packages.debian.org/devscripts`) que contiene: `debchange`,`debclean`,`release`,`build`,`depkg`,`debi`,`debc`,`dch`,`uupdate`, `uscan`, y, finalmente,`deblint`, una herramienta muy útil para ver si el paquete cumple estrictamente todos los requisitos de la política de Debian. El uso de estas herramientas es muy sencillo, por ejemplo, para incorporar cambios al fichero `debian/changelog` se puede ejecutar `dch texto_del_cambio`, si además se quiere que sea una nueva versión con `dch -n texto_del_cambio`, el programa añadirá automáticamente la cabecera y pie según el formato definido (indicando fecha, hora y desarrollador).

También el paquete debhelper[7] contiene un buen número de herramientas que pueden usarse para construir, de una manera más sencilla, el fichero `debian/rules`, automatizando tareas habituales: instalar ficheros, comprimirlos, arreglar los permisos, integrar el paquete con el sistema de menú de Debian, etc.. Todas las utilidades

[5] `http://www.es.debian.org/social\_contract`
[6] `http://packages.debian.org/debmake`
[7] `http://packages.debian.org/debhelper`

proporcionadas por este paquete comienzan con `dh_`, así tenemos: `dh_installdocs`, `dh_installlexamples`, `dh_checkroot`....

Y no se debe dejar de mencionar a cvs-buildpackage[8] que permite crear paquetes a partir de un repositorio CVS[9] (Concurrent Versions System, un sistema de control de versiones muy versátil y ampliamente utilizado).

7 El formato .deb

Los ficheros `.deb` generados por el procedimiento ya visto, no son sino una serie de ficheros encadenados con el programa `ar`, en total tres: `data.tar.gz`, `control.tar.gz` y `debian-binary`. Los dos primeros son, por un lado un tar.gz (data.tar.gz) con el árbol de directorios que se genera en `debian/tmp` y que se desempaquetará directamente sobre el raíz del disco duro en el momento de instalar, y por otro el directorio DEBIAN (control.tar.gz) que contiene muchos de los ficheros vistos en `debian/`, aunque algunos estarán modificados.

Es posible extraer estos por separado, el tar.gz con el comando `dpkg -x fichero.deb directorio_destino` y el DEBIAN con el comando `dpkg -c fichero.deb directorio_destino`. Aunque en realidad esto se puede hacer también con `ar -x fichero.deb`, lo que hace posible instalar un paquete Debian incluso en un sistema que no sepa nada de distribuciones, simplemente con tener la herramienta GNU `ar` ya instalada. También se puede construir un fichero .deb (es decir hacer el proceso inverso) con el programa `ar` o con `dpkg -build directorio` que creará el fichero `directorio.deb`.

8 Diferencias con otros sistemas

En realidad no se ha contado la política de Debian respecto a la instalación de paquetes, que define desde dónde deben colocarse los ficheros hasta qué modificaciones puede hacer un paquete a un sistema, o a través de qué métodos puede hacerlo (por ejemplo usando el sistema menu para incluir aplicaciones en los menús de todos los gestores de ventanas X). Se recomienda al lector que acuda a los punteros indicados para entrar en el detalle, sin embargo sí es interesante comentar algunas de las diferencias que hacen que Debian sobresalga por encima de otros sistemas:

- la base de datos del sistema de paquetes está en texto en claro, es posible arreglar problemas de corrupción a mano sin que el sistema se quede inutilizado si la base de datos queda corrompida.
- existen un buen número de herramientas para la gestión de paquetes, estando el diseño de éstos muy bien documentado.
- hay un fuerte seguimiento de dependencias, especialmente con la nueva herramienta de instalación de paquetes que apareció en Debian 2.0 llamada apt[10].
- Los paquetes se desempaquetan en un orden que minimiza el tiempo durante el cual no están disponibles, asimismo el sistema de paquetes garantiza que programas que se puedan hacer "daño" unos a otros no estén instalados simultáneamente en la misma máquina.
- la cooperación entre paquetes y el sistema se hace posible de varias formas: a través del paquete menu[11], definiendo un estándar de acceso a la documentación mediante el paquete doc-base[12] (con la documentación en HTML en http://localhost/doc) y de instalación de servidor (raíz del servidor y residencia de cgi's) lo que permite a los paquetes integrarse con el servidor local de web.
- su adherencia a los estándards es firme, no sólo existen estándards, sino que se cumplen, existiendo un seguimiento constante de que se cumpla la política definida (ver sino la pagina de Lintian[13], también disponible como paquete[14])
- permite la coexistencia de distintas versiones de una misma librería.
- es posible tener distintas versiones del mismo kernel, o compilar el kernel junto con los módulos fácilmente.

[8] `http://packages.debian.org/cvs-buildpackage`
[9] `http://www.loria.fr/~molli/cvs-index.html`
[10] `http://packages.debian.org/apt`
[11] `http://packages.debian.org/menu`
[12] `http://packages.debian.org/doc-base`
[13] `http://www.debian.org/lintian`
[14] `http://packages.debian.org/lintian`

Y se está trabajando en el uso posible de linuxconf. Habiéndose terminado ya selecciones prefabricadas de paquetes, de forma que un usuario pueda elegir cosas genéricas (desarrollo de web, juegos, desarrollo software...) en la instalación y obtener una selección de paquetes relevantes; para no tener que navegar por entre los 2500 paquetes disponibles en Debian 2.1.

Con todo esto y más, Debian demuestra que su sistema de paquetes es robusto y confiable, más aún que los de otras distribuciones. Esto, junto a la gran calidad y variedad de programas que acompañan a la distribución, y el ser un sistema abierto a todos aquellos que deseen colaborar (quizás el lector después de leer este artículo desee hacerlo) lo convierte en un sistema muy a tener en cuenta en el mundo de GNU/Linux.

9 Apéndice:Construcción del paquete hello

```
templar@root:/tmp/hello-1.3$ dpkg-buildpackage
dpkg-buildpackage: source package is hello
dpkg-buildpackage: source version is 1.3-13
dpkg-buildpackage: build architecture is i386
debian/rules clean
test -f hello.c -a -f debian/rules
rm -f build make -i distclean || make -f Makefile.in distclean
make[1]: Entering directory ‘/tmp/hello-1.3’
rm -f hello *.o core test.out hello.dvi hello.?? hello.??s rm -f
Makefile config.status
make[1]: Leaving directory ‘/tmp/hello-1.3’
rm -rf *~ debian/tmp debian/*~ debian/files*
dpkg-source -b hello-1.3
dpkg-source: building hello using existing hello_1.3.orig.tar.gz
dpkg-source: building hello in hello_1.3-13.diff.gz
dpkg-source: building hello using existing hello_1.3.orig.tar.gz
dpkg-source: building hello in hello_1.3-13.diff.gz
dpkg-source: building hello in hello_1.3-13.dsc
 debian/rules build
test -f hello.c -a -f debian/rules
./configure --prefix=/usr checking for gcc (...)
make[1]: Entering directory ‘/tmp/hello-1.3’ (...)
gcc -o hello hello.o version.o getopt.o getopt1.o
make[1]: Leaving directory ‘/tmp/hello-1.3’
touch build
 debian/rules binary t
est -f hello.c -a -f debian/rules
test root ="‘whoami‘"
test -f hello.c -a -f debian/rules
test -f hello.c -a -f debian/rules
rm -rf debian/tmp install -d debian/tmp debian/tmp/DEBIAN
install -d debian/tmp/usr/doc/hello
cp debian/{postinst,prerm} debian/tmp/DEBIAN/.
chmod +x debian/tmp/DEBIAN/{postinst,prerm}
make CFLAGS=-O2 LDFLAGS=-s INSTALL_PROGRAM=’install -c -s’ \
        prefix=debian/tmp/usr install
make[1]: Entering directory ‘/tmp/hello-1.3’ ./mkinstalldirs debian/tmp/usr/bin
debian/tmp/usr/info install -c -s hello debian/tmp/usr/bin/hello
/usr/bin/install -c -m 644 ./hello.info debian/tmp/usr/info/hello.info
make[1]: Leaving directory ‘/tmp/hello-1.3’ g
zip -9v debian/tmp/usr/info/*
cp debian/copyright debian/tmp/usr/doc/hello/.
cp debian/changelog debian/tmp/usr/doc/hello/changelog.Debian
cp ChangeLog
debian/tmp/usr/doc/hello/changelog
gzip -9v debian/tmp/usr/doc/hello/changelog{,.Debian}
```

```
dpkg-shlibdeps hello
dpkg-gencontrol chown -R root.root debian/tmp
chmod -R g-ws debian/tmp
dpkg --build debian/tmp ..  d
pkg-deb: building package ‘hello’ in ‘../hello_1.3-13_i386.deb’.
signfile hello_1.3-13.dsc
Pretty Good Privacy(tm) 2.6.2i - Public-key encryption for the masses.  (c)
1990-1995 Philip Zimmermann, Phil’s Pretty Good Software. 7 May 95
(...)
 dpkg-genchanges
dpkg-genchanges: not including original source code in upload
signfile hello_1.3-13_i386.changes
Pretty Good Privacy(tm) 2.6.2i - Public-key encryption for the masses.
(c) 1990-1995 Philip Zimmermann, Phil’s Pretty Good Software. 7 May 95 (...)
dpkg-buildpackage: diff-only upload (original source NOT included) ***
```

10 Apéndice: Los nombres de los paquete Debian

En Debian los nombre de los paquetes siguen una estructura estándar que es nombre+versión+arquitectura.deb. La arquitectura podrá ser *i386* (PCs con 386 o superior), *alpha*, *sparc*, *m68k*, *powerpc*, o *arm*, pero se está haciendo un gran esfuerzo por llevar a Debian a otras arquitecturas como e incluso a otros núcleos (cómo el *Hurd*). El número de versión es de la forma [epoca:]versión-upstream[-revisión-debian].

- *epoca*: Un entero generalmente pequeño, si no existe se asume que es 0. Se utiliza para soportar el cambio de sistemas de numeración de versiones que pueda hacer el autor original. Generalmente no se muestra.
- *version-upstream*: Ésta es la parte principal de la versión, se trata del número de versión del paquete original (upstream) del cual se ha hecho el fichero `.deb`. Normalmente se mantiene el formato usado por el autor original (aunque a veces pueda tener que ser modificado para que no existan conflictos), sólo puede tener los caracteres alfanuméricos y '+','-','.' o ':' y debe comenzar por un dígito.
- *revision-debian*: Ésta parte de la versión representa la versión de las modificaciones hechas al paquete para convertirlo en un paquete para Debian. Usa el mismo formato que el anterior (puede no existir, si el software ha sido creado específicamente para Debian).

Seguir este esquema es importante porque Debian lo usa para resolver conflictos y dependencias, que dependen, en muchos casos, de una versión determinada. Sólo con un esquema fijo puede `dpkg` saber si una versión es más nueva o más vieja que otra.

11 Apéndice: Fichero rules del paquete hello (traducido)

```
#!/usr/bin/make -f
# Ejemplo de fichero debian.rules - para GNU Hello (1.3)
# Copyright 1994,1995 por Ian Jackson.
# Te doy permiso perpetuo e ilimitado para copiar, modifiar y relicenciar este fichero,
# siempre y cuando no borres mi nombre de este fichero (Yo asevero mi derecho
# moral de paternidad bajo el Acta de Copyright, Diseño y Patentes de 1988)
# Este fichero puede necesitar de modificaciones extensas.

# Solía haber unos objetivos llamados 'source' y 'diff' en este
# fichero, y muchos paquetes también han tenido 'chanes' y
# 'dist'. Estas funciones han sido recogidas por dpkg-source,
# dpkg-genchanges y dpkg-buildpackage en una forma independiente del
# paquete, estos objetivos están, pues, obsoletos

package=hello

build:
```

```
        $(checkdir)
        ./configure --prefix=/usr
        $(MAKE) CFLAGS=-O2 LDFLAGS=
        touch build clean:
        $(checkdir)
        -rm -f build
        -$(MAKE) -i distclean || $(MAKE) -f Makefile.in distclean
        -rm -rf *~ debian/tmp debian/*~ debian/files*

binary-indep: checkroot build
        $(checkdir)
# No hay ningun fichero independiente de arquitectura generado por
# este paquete.  Si lo hubiera se haría aquí.

binary-arch: checkroot build
        $(checkdir)
        -rm -rf debian/tmp
        install -d debian/tmp debian/tmp/DEBIAN
        install -d debian/tmp/usr/doc/$(package)
        cp debian/{postinst,prerm} debian/tmp/DEBIAN/.
        chmod +x debian/tmp/DEBIAN/{postinst,prerm}
        $(MAKE) CFLAGS=-O2 LDFLAGS=-s INSTALL_PROGRAM='install -c -s' \
                prefix=debian/tmp/usr install
        gzip -9v debian/tmp/usr/info/*
        cp debian/copyright debian/tmp/usr/doc/$(package)/.
        cp debian/changelog
debian/tmp/usr/doc/$(package)/changelog.Debian
        cp ChangeLog debian/tmp/usr/doc/$(package)/changelog
        gzip -9v debian/tmp/usr/doc/$(package)/changelog{,.Debian}
        dpkg-shlibdeps hello
        dpkg-gencontrol
        chown -R root.root debian/tmp
        chmod -R g-ws debian/tmp
        dpkg --build debian/tmp ..

define checkdir
        test -f $(package).c -a -f debian/rules endef

# Esto de aquí abajo es bastante genérico

binary: binary-indep binary-arch

source diff:
        @echo >&2 'source and diff are obsolete - use dpkg-source -b';
false

checkroot:
        $(checkdir)
        test root = "`whoami`"

.PHONY: binary binary-arch binary-indep clean checkroot
```

12 Apéndice: Más información

Se puede encontrar más información del sistema de paquetes de Debian en el servidor de Debian, en `http://www.debian.org` (el mirror español es `http://www.es.debian.org`), también existen una serie de paquetes que facilitan documentación detallada sobre el sistema de paquetes, en Debian son: debian-policy[15] (política que se debe seguir para crear paquetes para Debian), y developers-reference[16] (información para aquellos que quieren convertirse en desarrolladores oficiales de Debian); aunque se puede encontrar mucha información en un sistema Debian instalado en `/usr/doc/dpkg` y `/usr/doc/debian`. Las listas de distribución[17] también son una fuente importante de información, se encuentran indexadas en el servidor de Debian[18], en general, la lista `debian-devel@lists.debian.org` trata todos los temas de importancia para los desarrolladores de Debian, también existe una lista para usuarios (debian-user) y para usuarios españoles (debian-user-spanish).

En las réplicas de la distribución de Debian también se puede encontrar más información en el subdirectorio projects[19].

13 Acerca de este artículo

Este artículo fue escrito por Javier Fernández-Sanguino Peña para la revista OpenResources.

La versión original de este artículo está disponible en `http://www.openresources.com/es/magazine/making-debian-packages/`, en la Revista OpenResources[20]

14 Cómo hacer un paquete Debian

Esta sección es la traducción del documento "Debian New Maintainers Guide" de Josip Rodin, e incluida en el paquete Debian 'maint-guide-es'

14.1 Empezando "de la Forma Correcta"

Este documento tratará de describir cómo se construye un paquete Debian GNU/Linux para el usuario normal de Debian (y 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 formato de paquetes de Debian, algunas veces necesitará instalar programas que no estén así. Puede preguntarse cómo hará vd. sus propios paquetes y que quizás esta sea una tarea demasiada difícil. Bueno, si es vd. un novato en Linux, sí es duro, pero si eres todo un experto, no deberías estar leyendo esto ahora mismo. :–) Necesitas saber algo sobre programación en Unix, pero desde luego no tienes que ser un maestro.

14.1.1 Programas que necesitas para el desarrollo

Antes de empezar nada, deberías asegurarte de instalar adecuadamente algunos paquetes con `dpkg(8)` (esto es, 'dpkg –i paquete', o a través de alguno de sus interfaces como dselect y apt). Este documento fue escrito mientras la 'hamm' 2.0 y 'slink' 2.1 eran las distribuciones estables oficiales de Debian, por ello los paquetes que se indicarán aquí son aquellos de 2.1.

Los siguientes paquetes vienen en una instalación estándar de Debian 2.1, así que probablemente ya los tenga. Aun así, debería comprobarlo con 'dpkg –s paquete'.

- *binutils* – estos programas se usan para ensamlar y enlazar ficheros objetos – aquellos que componen los programas (vea 'info binutils')
- *cpp* – el preprocessador de C. (vea `cpp(1)`)
- *cpio* – este es un programa archivador como tar o zip. (vea `cpio(1)`)
- *dpkg–dev* – este paquete contiene las herramientas necesarias para desempaquetar, construir y enviar paquetes fuente de Debian. También contiene los manuales de empaquetamiento y de las tripas de dpkg (dpkg–internas, n. del t.). (vea `dpkg-source(1)`)

[15] `http://packages.debian.org/debian-policy`
[16] `http://packages.debian.org/developers-reference`
[17] `http://www.es.debian.org/MailingLists`
[18] `http://www.debian.org/Lists-Archives`
[19] `ftp://ftp.es.debian.org/debian/projects`
[20] `http://www.openresources.com/es/magazine/`

- *file* – este útil programa puede determinar de qué tipo es un fichero. (vea `file(1)`)
- *fileutils* – las utilidades esenciales de Linux, como ls, chmod, rm y otras. (vea 'info —file /usr/info/fileutils.info.gz')
- *gcc* – el compilador de C de GNU. La mayor parte de los programas de Linux están escritos en C. Si su programa está escrito en algún otro lenguaje de programación como C++, Fortran o Pascal, debería instalar g++, g77, o gpc, respectivamente. (lea `gcc(1)`, `g++(1)`, `g77(1)`, `gpc(1)`)
- *libc6–dev* – las librerías y cabeceras de fichero de C que gcc necesita para enlazar y crear ficheros objeto. Aunque alugnos programas recomienda y/o usan libc5, le sugiero que utilize la nueva versión (libc6). (vea 'info libc')
- *make* – habitualmente la creación de un programa tiene varios pasos. En lugar de ejecutar los mismos comandos una y otra vez, puede utilizar este programa para automatizar el proceso, creando 'Makefiles'. Algunos programas también usan imake y xmkmf, programas para generar Makefiles de un conjunto de funcioens macro. Algunos programas más nuevos usan scripts de configuración y Makefiles con la ayuda de programas como autoconf y automake, así que puede necesitarlos también. (vea 'info make', `imake(1)`, `xmkmf(1)`, `autoconf(1)`, `automake(1)`)
- *patch* – esta utilidad es muy útil ya que permite coger el fichero que contiene un listado de diferencias (producido por el programa diff) y aplicarselas al fichero original, produciendo una versión "parcheada". (vea `patch(1)`)
- *perl* – este es uno de los lenguajes interpretados para hacer scripts más usados en los sistemas un*x de hoy, comúnmente referido como la languages on today's un*x systems, often referred to as "Sierra Mecánica Suiza de Unix". (vea `perl(1)`)

De la sección 'devel' de la distribución posiblemente necesite instalar esto usted mismo:

- *fakeroot* o *libtricks* – estos le permiten emular ser root (superusuario, n. del t.) lo cual es necesario para ciertas partes del proceso de construcción. (vea `fakeroot(1)`)
- *lintian* – este paquete le indica muchos de los errores comunes después de construir un paquete, y explica los errores. Hace falta tener instalado diffstat también, una pequeña utilidad que crea histogramas a partir de la salida de diff. (vea `lintian(1)`, `diffstat(1)`, /usr/doc/lintian/lintian.html/index.html)

Y de la sección 'utils' necesitará obtener estos paquetes:

- *dh–make* y *debhelper* – dh–make es necesario para crear el esqueleto de nuestro paquete de ejemplo, y usará algunas de las herramientas de debhelper para crear paquetes. No son necesarios para la creación de paquetes, pero se recomienda **encarecidamente** a nuevos desarrolladores. Hace que el proceso sea más fácil de empezar y controlar más tarde. (vea `dh_make(1)`, `debhelper(1)`, /usr/doc/debhelper/README)
- *devscripts* – este paquete contiene algunos scripts bonitos y útiles que pueden ser de ayuda a los desarrolladores, pero tampoco son necesarios para construir paquetes. (vea /usr/doc/devscripts/README.debian.gz)
- *debmake* – este paquete contiene algunos programas que funcionan de manera similar a dh–make, pero su uso específico **no** está cubierto en este documento. Lea, si lo desea, el manual de Debmake (`http://www.debian.org/~jaldhar/`).

Por último, estos paquetes de *gran importancia* están en la sección 'doc' de la distribución:

- *debian–policy* – incluye la estructura y contenidos del archivo, ciertas notas sobre diseño del SO, el Estandar de la Estructura del Sistema de Ficheros de Linux (Linux Filesystem Structure Standard, n. del t), y, lo más importante (para usted) es que describe los requisitos que debe satisfacer cada paquete para ser incluido en la distribución. (vea /usr/doc/debian–policy/policy.html/index.html)
- *developers–reference* – para todos los temas no específicamente relacionados con los detalles técnicos de cómo empaquetar, como la estructura del archivo, como renombrar, abandonar, coger paquetes, cómo hacer NMUs (Non–Maintainer Uploads, o envíos por personas distintos del desarrollador, n. del t.), como gestionar los bugs (errores, n. del t.) y cómo y cuando enviar etc. (lea /usr/doc/developers–reference/developers–reference.html/index.html)

- *packaging–manual* – describe los aspectos técnicos de cómo crear paquetes binarios y fuentes en Debian. (lea /usr/doc/packaging–manual/packaging.html/index.html)

También necesitará el paquete de criptografía `pgp(1)` para *firmar* digitalmente su paquete. Esto es especialmente importante si quiere distribuir su paquete a otras personas (y hará precisamente esto cuando si su paquete se incluye en la distribución de Debian). Sin embargo, debido a las leyes de exportación de los EEUU, no puede simplemente obtenerlo de su servidor de FTP de Debian más cercano. Pero su servidor de FTP tendrá un fichero llamado README.non–us, que le indicará cómo obtener una copia de pgp (pista: ftp://nonus.debian.org/debian–non–US/).

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

14.1.2 Más información

Usted puede construir dos tipos de paquetes, fuentes y binarios. Un paquete fuente contiene el código que puede compilar en un programa. Un paquete binario contiene sólo el programa terminado. ¡No mezcle los términos como fuentes de un programa y el paquete fuente de un programa! Por favor lea los otros manuales si necesita más detalles sobre terminología.

Debian usa el término 'desarrollador' para la persona que hace paquetes, 'autor' para la persona que hizo el programa, y 'desarrollador fuente' ('upstream maintainer', n. del t.) para la persona que actualmenete mantiene el programa. Generalmente el autor y el desarrollador fuente son la misma persona – y algunas veces incluso el desarrollador es el mismo. Si hace vd. un programa, y quiere incluirlo en Debian, tiene total libertad para solicitar convertirse en desarrollador.

Después de construir su paquete (o mientras lo hace), deberá convertirse en un desarrollador oficial de Debian si desea que su programa entre en la próxima distribución (si el programa es útil, ¿por qué no?). Este proceso se explica en la Referencia del Desarrollador, por favor, leala.

14.2 Primeros pasos

Mientras que la documentación en el Rincón del Desarrollador (`http://www.debian.org/devel/`) no está tan clara sobre dónde y cómo debería empezar un desarrollador su trabajo, este documento explicara cada pequeño (y algunas veces incluso irrelevante) detalle, para ayudarle crear ese primer paquete, y ganar alguna experiencia en la construcción de nuevas versiones de éste y quizás otros paquetes más adelante.

14.2.1 Elija su programa.

Porbablemente haya escogido ya el paquete que desea construir, pero aquí hay algunos punteros para los novatos:

- compruebe si el paquete ya está en la distribución. Si usa la distribución 'estable', quizás sea mejor que vaya a la página de búsqueda de paquetes (`http://www.debian.org/distrib/packages.html`). Si usa la distribución 'inestable' **actual**, compruebe esto con los comandos:

  ```
  dpkg -s programa
  dpkg -l '*programa*'
  ```

- consulte la página WNPP (`http://www.debian.org/doc/prospective-packages.html`) para ver si alguna otra persona está construyendo ese mismo programa. Si es así, contacte el desarrollador actual si creo que lo necesita mantener. Sino, intente buscar otro programa interesante que nadie mantenga.

- el programa **debe** tener una licencia, si es posible, libre en el sentido marcado por las Guías del Software Libre de Debian (`http://www.debian.org/social_contract.html#guidelines`). Si no sigue una de estas reglas, aún puede incluirse en las secciones 'contrib' o 'non–free' de Debian. Si no está seguro sobre en qué lugar debería ir, pregunte en `<debian-legal@lists.debian.org>`.

- el programa **no** debería ejecutarse con setuid root, o aún mejor: no debería estar 'setuid' ni 'setgid' a nada.

- el programa no debería ser un demonio, o algo que va en los directorios */sbin.

- el programa debería ser binarios ejecutables, no intente librerías aún.
- debería estar bien documentado, o al menos legible (para cualquiera)
- debería contactar el autor/es del programa para comprobar si están de acuerdo con que se empaquete. Esto es importante para consultar al autor/es sobre el programa en caso de que haya problemas específicos, así que no intente empaquetar programas que no estén mantenidos.
- y por último pero no menos importante, debería saber cómo funciona, y haberlo utilizado por algún tiempo.

Por supuesto, esta lista es para tomar medidas de seguridad, y con la intención de salvarle de usuarios enfurecidos is hace algo mal con algún demonio de ftp setuid... Pero cuando tenga más experiencia en empaquetar cosas, podrá hacer este tipo de paquetes, pero incluso los desarrolladores más experimentados preguntan en la lista de distribución de debian–devel cuando tienen dudas. Y la gente allí le ayudara gustosamente.

Para más ayuda sobre esto, lea la Referencia del Desarrollador

14.2.2 Obtenga el programa, y pruebelo.

La primera cosa a hacer es encontrar y descargar el paquete original. Supongo que ya tiene el código fuente que obtuvo de la página del autor. Las fuentes de Linux generalmente vienen en el formato tar/gzip, con extensión `.tar.gz` o `.tgz`, y generalmente contienen un subdirectorio llamado programa–versión con todas las fuentes en él. Si su programa viene en otro tipo de archivo (como por ejemplo, el fichero termina con `.Z` o `.zip`), descomprimal con las herramientas adecuadas, o pregunte en debian–mentors si no está seguro de cómo se puede desempaquetar correctamente (pista: pruebe 'file archivo.extension').

Como ejemplo, usaré el programa conocido como 'gentoo', un gestor de ficheros de X11 con GTK+.

Cree un subdirectorio bajo /usr/local/src que tenga el mismo nombre que su programa (/usr/local/src/gentoo en este caso). Y mueva a él el archivo que ha descargado, y descomprimalo de la siguiente forma: 'tar –xzf gentoo–0.9.12.tar.gz'. Este proceso (algo largo) no dará ninguna salida (excepto si hay algún error, con lo que tendrá que bajarse de nuevo el archivo o comprobar que es un fichero tar/gzip), pero tendrá las fuentes desempaquetadas en un subdirectorio llamado 'gentoo–0.9.12' en /usr/local/src/gentoo.

Muevase a ese directorio y lea **en profundidad** la documentación que encuentre. Está generalmente en ficheros que se llaman README*, INSTALL*, *.lsm o *.html. Allí encontrará instrucciones de cómo compilar e instalar el programa (generalmente para el directorio /usr/local/bin).

El proceso varia de programas a programas, pero gran parte de los programas modernos vienen con un script 'configure' que configura las fuentes para su sistema y se asegura de que su sistema está en condiciones de compilarlo. Después de configurarlo (con './configure'), los programas generalmente se compilan con 'make', y se instalarn en sus directorios de destino ejecutando 'make install'.

Así que compile, instale y pruebe el programa, asegurese de que funciona bien y que no rompe nada más mientras está instalándose o ejecutándose.

14.2.3 Cosas antes de 'dh_make'

Para construir correctamente el paquete, debería mover el directorio de fuentes a <nombre_de_paquete>–<versión>. Como puede ver, el programa de ejemplo no lo necesita, pero quizás su programa sí. También, ponga el nombre en minúsculas si no o está ya. Si consiste de una o más palabras, contraialas a una palabra o haga una abreviatura. Por ejemplo, el paquete del programa "el editor para X de Javi"se podría llamar javiedx o jle4x, o lo que decida, siempre y cuando no se exceda de unos límites razonables, como 15 caracteres.

También compruebe la versión exacta del programa (¡no del paquete!). Si el programa no está numerado con versiones del estilo de X.Y.Z, pero con fecha de lanzamiento, es usted libre de utilizar la fecha (si la fecha es el 19 de diciembre de 1998, utilize la abreviatura norteamericana 19981219) cmomo número de versión. Aún así habrá algunos que ni siquiera estén numerados, en cualqueir caso debe contactar con el 'desarrollador fuente' para ver si tienen algún otro sistema de seguimiento de revisiones.

Antes de empezar con el proceso dh_make, debería poner en su variable de entorno $EMAIL si dirección de correo, y lo hara haciendo algo como esto en su shell (esto es para basj).

```
export EMAIL=usuario.login@algun_lugar.net
```

14.2.4 Ejecutando 'dh_make'

Asegúrese que está en el directorio donde están las fuentes del programa, y ejecute lo siguiente:

```
dh_make
```

Saldrá alguna información. Le preguntará qué tipo de paquete desea crear. Gentoo es un solo paquete de binarios – crea sólo un binario, y, por tanto, sólo un fichero .deb – así que seleccionaremos la primera opción, con la tecla 's'. Como nuevo desarrollador, está desaconsejado crear paquetes multibinarios, o librerías, como se explicó antes.

Tenga en cuenta que debería ejecutar dh_make **sólo una vez**, y que no se comportará correctamente si lo hace otra vez en el mismo directorio, ya "debianizado". Esto también significa que usará un método distinto para crear una nueva revisión o una nueva versión de su paquete en el futuro. Lea más sobre esto más adelante en el texto.

14.3 Modificando las fuentes.

Cuando dh_make termina, y haya ajustado el Makefile del propio programa, podrá hacer 'cd ..' para ver el nuevo directorio que ha creado, que se llama 'gentoo–0.9.12.orig'. Contiene el código fuente original que de ahora en adelante permanecerá intacto. El directorio 'gentoo–0.9.12' todavía existe, es allí donde hará las modificaciones.

Normalmente, 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 del usuario). Esto significa que tiene que mirar el Makefile the gentoo. Este es el script `make(1)` que usará para automatizar la creación de éste programa. Para más detalles sobre Makefiles, mire en 'el fichero 'rules'.' en la página 18.

Tenga en cuenta que no hay espacio aquí para entrar en *todos* los detalles respecto a los arreglos, pero aquí hay algunos de los problemas frecuentes a los que se enfrenta uno.

14.3.1 El problema de $DESTDIR

```
# ¿Dónde poner el binario en  'make install'?
BIN     = /usr/local/bin
# ¿Dónde poner los iconos en 'make install'? Nota: si cambia esto,
# gentoo no encontrará los iconos cuando arranque. Deberá cambiar
# el path de iconos de gentoo (en la ventana de configuración:
# "Paths") para que funcione.
ICONS   = /usr/local/lib/gentoo/
```

Antes de esto debería insertar dos nuevas líneas que dicen:

```
# Editado para Debian GNU/Linux.
DESTDIR =
```

porque el proceso de construcción lo necesita (explicado más tarde).

Después el Makefile menciona la localización del binario final. Sólo necesita cambiar esto:

```
# ¿Dónde poner el binario en  'make install'?
BIN     = $(DESTDIR)/usr/X11R6/bin
```

¿Pero por qué en este directorio y no en otro? Porque Debian tiene unas reglas definidias de dónde deberían estar instalados los programas. Están especificas en el Linux Filesystem Structure Standard (/usr/doc/debian–policy/fsstnd) [Estándar de Linux de la Estructura de los Sistemas de Ficheros, n. del t.]. Así, deberíamos instalar el binario en /usr/X11R6/bin en lugar de /usr/local/bin, y la página de manual (no existe aquí, pero casi todos los programas tienen una así que haremos una después) en /usr/man/man1 en lugar de /usr/local/man/man1.

Después de esto tenemos un situación un poco más complicada. Si cambia la línea a:

```
ICONS   = $(DESTDIR)/usr/share/gentoo/
```

que estará dentro de la política de Debian, deberá editar algunos ficheros de fuentes reales de C. Pero, ¿dónde buscar? Puede probar a encontrarlos usando:

```
grep -n usr/local/lib *.[ch]
```

(en cada subdirectorio que contiene ficheros .c y .h). Grep dirá el nombre del fichero y la línea, cuando encuentra una ocurrencia. Ahora edite esos ficheros y cambie en esas líneas usr/local/lib con usr/share – y ya está. Simplemetne remplaze usr/local/lib por su localización, y sea muy cuidadoso para no mezclar el resto del código, si no sabe mucho sobre cómo programas en C. :–)

Después de esto debería encontrar el objetivo 'install' (busque una línea que comienze por 'install:') y renombre todas las referencias a directorios distintos de los definidos al comienzo del Makefil. En este caso, hace esto y limpia las cosas un poco, anteriormente el objetivo install decía:

```
# ----------------------------------------- Installation

# ¡Debe ser superusuario para hacer esto!
install:        gentoo-target
                install ./gentoo $(BIN)
                install icons $(ICONS)
                install gentoorc-example $(HOME)/.gentoorc
```

Después de su cambio dice:

```
# ----------------------------------------- Installation

# You're going to have to be root to do this!
install:        gentoo-target
                install -d $(BIN) $(ICONS) $(DESTDIR)/etc
                install ./gentoo $(BIN)
                install -m644 icons/* $(ICONS)
                install -m644 gentoorc-example $(DESTDIR)/etc/gentoorc
                install -d $(DESTDIR)/usr/doc/gentoo/html
                cp -a docs/* $(DESTDIR)/usr/doc/gentoo/html
```

Un lector atento se dará cuenta de que he cambiado 'gentoo' a 'gentoo–target' en la línea 'install:'. Esto es para arreglar un fallo en el programa. :–)

Siempre que haga cambios que no estén específicamente relacionados con el paquete Debian, asegurese de que los envía al desarrollador original para que éste los incluya en la próxima revisión del programa.

14.3.2 Librerias diferentes.

Hay un problema comúna: las librerías son generalmente diferentes de plataforma a plataform. Por ejemplo, un Makefile puede contener una referencia a una librería que no exista en Debian. En este caso, se necesita cambiarla a una librería que sí existe en Debian y sirve para el mismo propósito. La mejor forma es comentar (*¡no borrar!*) esas líneas porque puede que haber otras personas que intenten compilar en diferentes plataformas y éstas les pueden dar algunas pistas de dónde puede estar el problema.

Así, si hay una línea en el Makefil de su programa que dice algo como lo siguiente (y su programa no compila):

```
LIBS = -lcurses -lsomething -lsomethingelse
```

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

```
LIBS = -lncurses -lsomething -lsomethingelse
#LIBS = -lcurses -lsomething -lsomethingelse
```

14.4 Los ficheros debian/control y debian/rules.

Ahora hay un nuevo subdirectorio en gentoo–0.9.12, que se llama 'debian'. Hay bastantes ficheros en este directorio. Empezaremos editando éstos para adaptar el comportamiento del paquete. La parte más importante es modificar los ficheros 'control' y 'rules' (reglas, n. del t.).

14.4.1 El fichero ‘control’.

Este fichero contiene varios valores que dpkg y dselect 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 <jrodin@jagor.srce.hr>
5 Standards-Version: 2.4.0.0
6
7 Package: gentoo
8 Architecture: any
9 Depends: ${shlibs:Depends}
10 Description: Missing
11 Missing
```

(He añadido los números de línea.)

Las lineas 1 a 5 son la información de contol del paquete fuente. La línea 1 es el nombre del paquete fuente.

La línea 2 es la sección de la distribución en la que va este paquete. Como puede haber observado, Debian está divido en secciones: main (principal, n. del t.) (el software libre), non–free (el software que no es libre) y contrib (el software libre que depende del software no libre). Por debajo de éstas, hay subsecciones lógicas que describen brevemente qué paquetes tienen. Así tenemos ’admin’ para programas que son sólo para los administradores, ’base’ para las herramientas básicas, ’devel’ para las herramientas de los programadores, ’doc’ para documentación, ’libs’ para librerías, ’mail’ para lectores y demonios de correo, ’net’ para aplicaciones de red y demonios, ’x11’ para programas específicos de X11 y muchas otras.

Lo cambiaremos, pues, a x11.

La línea 3 describe cómo de importante es para el usuario la instalación de éste paquete. La sección y prioridad son actualmente sólo usadas por dselect cuando ordena paquetes y selecciona valores por defecto, y pueden ser modificadas (y generalmente lo serán) por nuestros administradores del FTP. Lea el manual de Política de Debian para una guía de qué valor deben tener estos campos.

Como es un paquete de prioridad normal, lo dejaremos con prioridad ’optional’ (opccional, n. del t.).

La línea 4 es el nombre y correo electrónico del desarrollador.

La línea 5 es la versión de los estándares de la Política de Debian que sigue este paquete (dos versiones importantes del paquete debian–policy instalado).

La línea 7 es el nombre del paquete binario.

La línea 8 describe la arquitectura de CPU para la que se compiló el paquete. Podemos dejar ésta como ’any’ (cualquiera, n. del. t), ya que `dpkg-gencontrol(1)` lo rellenará con el valor apropiado cuando se compile este paquete.

La línea 9 muestra una de las más poderosas posibilidades del sistema de paquetes de Debian. Los paquetes se pueden relacionar unos con otros de diversas formas. Aprte 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.). Ésto es lo que significan:

- Depends:

  `dpkg(8)` y `dselect(8)` no se instalará el programa a menos que los paquetes de los que depende estén instalados. Use esto si su programa no funcionará de ninguna forma a menos que esté disponible un paquete determinado.

- Recommends:

  Dselect no le permitirá instalar el paquete a menos que los paquetes que recomienda estén instalados. Dpkg, sin embargo, sí le dejará. Use esto para paquetes que no son estrictamente necesarios pero generalmente son usados con su programa.

- Suggests:

  Cuando un usuario instale el programa, dselect le sugerirá instalar cualquier paquete que éste sugiera. Dpkg no hace caso a esto. Utilize esto para paquetes que funcionarán bien con su programa pero no son necesarios en absoluto.

- Pre–Depends:

  Esto es más fuerte que 'Depends'. Dpkg y dselect no instalarán el paquete a menos que el paquete del que pre–dependa esté instalado *y correctamente configurado.* Utilize esto **muy** poco y sólo después de haberlo discutido en la lista de distribución de debian–devel. En resumidas cuentas: no lo utilize en absoluto. :–>

- Conflicts:

  Dpkg y dselect no instalarán su programa hasta que todos los paquetes con los que entra en conflicto hayan sido eliminados.

- Provides:

  Para algunos tipos determinados de paquetes dónde hay múltiples alternativas se han definido nombres virtuales. Puede obtener la lista completa en el fichero /usr/doc/debian–policy/virtual–package–names–list.text.gz. Use esto si su pograma provee las funciones de un paquete virtual existente.

- Replaces:

  Use esto si su programa reemplaza ficheros de otro paquete, o reemplaza totalmente otro paquete (generalmente se usa conjuntamente con 'Conflicts':). Dpkg y dselect eliminarán los ficheros de los paquetes indicados antes de instalar el suyo.

Todos estos campos tienen una sintaxis uniforme. Tienen 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 aplicabilidad a versiones determinadas de cada paquete nombrado. Esto se hace entre paréntesis para cada nombre de paquete individual; los paréntesis deberían contener una relación de la siguiente lista seguida por un número de versión. Las relaciones permitidas son <<, <=, =, >= y >> estrictamente para anterior, anterior o igual, exáctamente igual, posterior o igual o estríctamente posterior, respectivamente.

La última funcionalidad que quiero enseñarle es $(shlibs:Depends). Ésta se sustituirá automáticamente por `dh_shlibdeps(1)` (ver más adelante) por los nombres de cualquier librería compartida, como libc6 o xlib6g, que use su programa, así que no necesita especificar esto vd. mismo. Habiendo dicho todo esto, puede dejar la línea 9 exáctamente como está ahora.

La línea 10 es donde va la lista de sugereencias. Aquí sólo es 'menu', porque gentoo debería estar en los menús del gestor de ventanas X11. Esto se controla también por el fichero debian/menu. Lea `menufile(5)`, y `update-menus(1)`.

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 50 caracteres. Cambiaré esto a "A fully GUI configurable GTK+ file manager"("La GUI a un gestor de ficheros GTL+ completamentae configurable").

La línea 12 es donde va la descripción larga del paquete. Debería ser al menos un párrafo que da más detalles del paquete. La Columna 1 de cada línea debería estar vacía. No puede haber líneas en blanco, pero puede poner un . (punto) en una columna para simularlo. También debe haber no más de una línea en blanco después de la descrición completa.

Aquí está el fichero de control actualizado:

```
1  Source: gentoo
2  Section: x11
3  Priority: optional
4  Maintainer: Josip Rodin <jrodin@jagor.srce.hr>
5  Standards-Version: 2.5.0
6
7  Package: gentoo
8  Architecture: any
9  Depends: ${shlibs:Depends}
10 Suggests: menu (>= 1.5)
11 Description: A fully GUI configurable GTK+ file manager
12 gentoo is a file manager for Linux written from scratch in pure C. It
13 uses the GTK+ toolkit for all of its interface needs. gentoo provides
14 100% GUI configurability; no need to edit config files by hand and re-
15 start the program. gentoo supports identifying the type of various
```

```
files (using extension, regular expressions, or the 'file' command),
and can display files of different types with different colors and icons.
.
gentoo borrows some of its look and feel from the classic Amiga file
manager "Directory OPUS" (written by Jonathan Potter).
```

14.4.2 el fichero 'rules'.

Ahora volvemos al directorio 'debian' para mirar las reglas que `dpkg-buildpackage(1)` utilizará para crear el paquete. Este fichero es en realidad otro Makefile, ya que es ejecutado con 'make –f', pero diferente al que viene en las fuentes originales.

Cada fichero 'rules' (de reglas, n. del t.), como muchos otros Makefiles, consisten de varias reglas sobre cómo compilar las fuentes. Las reglas consisten en objetivos: ficheros o nombres de accioens que se deben llevar a cabo (por ejemplo, 'build:' o 'install:'). Las reglas que quiere ejecutar deberían llamarse como los argumentos de comandos (por ejemplo, 'rules build' o 'rules install'). Después del nombre del objetivo, puede nombrar las dependencias, programas o ficheros de los que la regla depende. Después de esto hay un cualquier número de instrucciones (¡que empiezan con <tab>!), hasta que se llega a una línea en blanco, ahí empieza otra regla. Los comentarios empiezan con almohadillas ('#'), y terminan con el fin de la línea. Puede llamar a las reglas desde otra regla o desde la línea de órdenes (esto es 'debian/rules clean').

Probablemente ya se haya perdido, pero todo quedará más claro después de ver un fichero 'rules' que dh_make pone por defecto. Debería también leer la entrada de 'make' en info para más información.

```
#!/usr/bin/make -f
# Made with the aid of dh_make, by Craig Small
# Sample debian/rules that uses debhelper. GNU copyright 1997 by Joey Hess.
# Some lines taken from debmake, by Christoph Lameter.
# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1
build: build-stamp
build-stamp:
dh_testdir
# Add here commands to compile the package.
$(MAKE)
touch build-stamp
clean:
dh_testdir
dh_testroot
rm -f build-stamp install-stamp
# Add here commands to clean up after the build process.
-$(MAKE) clean
dh_clean
install: install-stamp
install-stamp: build-stamp
dh_testdir
dh_testroot
dh_clean -k
dh_installdirs
# Add here commands to install the package into debian/tmp.
```

```
$(MAKE) install DESTDIR=`pwd`/debian/tmp

touch install-stamp

# Build architecture-independent files here.
binary-indep: build install
# We have nothing to do by default.

# Build architecture-dependent files here.
binary-arch: build install
# dh_testversion
dh_testdir
dh_testroot
dh_installdocs
dh_installexamples
dh_installmenu
# dh_installemacsen
# dh_installinit
dh_installcron
dh_installmanpages
# dh_undocumented
dh_installchangelogs
dh_strip
dh_compress
dh_fixperms
dh_suidregister
dh_installdeb
dh_shlibdeps
dh_gencontrol
# dh_makeshlibs
dh_md5sums
dh_builddeb

source diff:
@echo >&2 'source and diff are obsolete - use dpkg-source -b'; false

binary: binary-indep binary-arch
.PHONY: build clean binary-indep binary-arch binary
```

(N. del T.: se traduce el fichero de reglas, dh_make sólo lo ofrece en inglés)

```
#!/usr/bin/make -f
# Creado con la ayuda de dh_make, por Craig Small
# Fichero de ejemplo debian/rules que usa debhelper. Copyright GNU 1997 por Joey Hess.
# Algunas líneas son de debmake, por Christoph Lameter.

# Quitele el comentario para activar el modo verboso
#export DH_VERBOSE=1

build: build-stamp
build-stamp:
dh_testdir


# Añada aquí los comandos para compilar el paquete.
$(MAKE)

touch build-stamp
```

```

clean:
dh_testdir
dh_testroot
rm -f build-stamp install-stamp

# Añada aquí los comandos para limpiar después del proceso de creación.
-$(MAKE) clean

dh_clean

install: install-stamp
install-stamp: build-stamp
dh_testdir
dh_testroot
dh_clean -k
dh_installdirs

# Añada aquí los comandos para instalar el paquete en debian/tmp.
$(MAKE) install DESTDIR=‘pwd‘/debian/tmp

touch install-stamp

# Construir los ficheros independientes de arquitectura aquí
binary-indep: build install
# Por defecto no se hace nada.

# Construir los ficheros dependientes de arquitectura aquí.
binary-arch: build install
# dh_testversion
dh_testdir
dh_testroot
dh_installdocs
dh_installexamples
dh_installmenu
# dh_installemacsen
# dh_installinit
dh_installcron
dh_installmanpages
# dh_undocumented
dh_installchangelogs
dh_strip
dh_compress
dh_fixperms
dh_suidregister
dh_installdeb
dh_shlibdeps
dh_gencontrol
# dh_makeshlibs
dh_md5sums
dh_builddeb

source diff:
@echo >&2 ’source y diff están obsoletos - use dpkg-source -b’; false

binary: binary-indep binary-arch
.PHONY: build clean binary-indep binary-arch binary
```

Probablemente esté familiarizado con líneas como la 1 de scripts hechos en shell o perl. Esto significa que el fichero debe ejecutarse con make. Las líneas vacías se ignoran. Las líneas que comienzan con '#' (almohadilla) se tratan como comentarios y pueden también ignorarse.

Las líneas 9 a la 17 describen la regla de construcción (en inglés: build, n. del t.) y su "hijo" 'build–stamp' que ejecuta el Makefile del propio programa para compilarlo.

Las cosas rara vez funcionan perfectamente la primera vez, así que se especifica la regla de limpieza (del inglés 'clean', n. del t.) en las líneas 18–26 que limpian cualquier resto innecesario dejado de intentos previos fallidos.

El proceso de instalación, la regla 'install', comienza en la línea 29. En la línea 34, todos los directorios necesarios se crean en el directorio 'debian'. La línea 37 llama al objetivo de instalación del Makefile de gentoo – e instala en el directorio debian/tmp – es por esto que específicamos como raíz del directorio de instalación $(DESTDIR) en el Makefile de gentoo.

Como sugiere el comentario, la regla 'binary–indep' en las líneas 41–43 se usa para construir paquetes independientes de arquitectura, pero aquí no hay nada.

Lo siguiente es la regla 'binary–arch', en las líneas 46 a 68, en la que ejecutamos varias utilidades diversas del paquete debhelper que nos permiten hacer operaciones variads en nuestro paquete para que cumpla la política de Debian.

Los nombres comienza con dh_ y a continuación se indica lo que realmente hace cada pequeña utilidad:

- `dh_testdir(1)` comprueba que estás en el directorio correcto (/usr/local/gentoo/gentoo–0.9.12/),
- `dh_testroot(1)` comprueba que tienes permisos de superusuario (root, n. del t.),
- `dh_installdirs(1)` crea los directorios que se mencionan en el fichero 'dirs' [no existe aquí],
- `dh_installdocs(1)` copia la documentación al directorio debian/tmp/usr/doc/gentoo,
- `dh_installmenu(1)` copia el fichero 'menu' a debian/tmp/usr/lib/menu/gentoo,
- `dh_installmanpages(1)` copia las páginas de manual y las enlaza correctamente,
- `dh_installchangelogs(1)` copia los ficheros 'changelogs' (registros de cambios, n. del t.) en el directorio debian/tmp/usr/doc/gentoo,
- `dh_installinit(1)` copia los scripts init.d [aquí no tenemos ninguno],
- `dh_installcron(1)` copia los scripts de crontab a debian/tmp/etc/cron.* [aquí no hay ninguno],
- `dh_installexamples(1)` copia los ficheros de ejemplos a debian/tmp/usr/doc/gentoo/examples [aquí no hay ninguno],
- `dh_strip(1)` elimina las cabeceras de depuración de los ficheros ejecutables para hacerlos más pequeños,
- `dh_compress(1)` comprime con 'gzip¡ las páginas de manual y los ficheros de documentación que sean más grandes de 4 kb,
- `dh_fixperms(1)` comprueba y arregla permisos no válidos en el directorio debian/tmp,
- `dh_suidregister(1)` adapta los ficheros para que se registren los ejecutables con 'setuid' con `suidregister(8)` [aquí no hay ninguno],
- `dh_installdeb(1)` copia los ficheros relativos al sistema de paquetes en el directorio debian/tmp,
- `dh_shlibdeps(1)` calcula las dependencias de los ejecutables,
- `dh_gencontrol(1)` genera e instala el fichero de control,
- `dh_makeshlibs(1)` genera el fichero de dependecias con librerías compartidas [aquí no existe],
- `dh_md5sums(1)` genera las sumas de chequeo MD5 , y finalmente,
- `dh_builddeb(1)` construye el paquete.

Cada uno de estos scripts dh_* tiene su propia página de manual, leala para más información. Hay otros scripts con la misma nomenclatura (dh_*) que no se han mencionado aquí, pero puede necesitar, lea la documentación de debhelper.

Las líneas 70 a la 74 son sólo algunas necesidades sobre las que puede leer en el mual de make. Por ahora, no es importante conocerlas.

La parte importante sobre el fichero de reglas creado por dh_make es que sólo es una sugerencia. Funcionará para paquetes simples pero para los más complicados no se asuste y modifique, borre o añada a éste para ajustarse a sus necesidades. Esto se aplica de forma especial a las secciones binary–arch, donde debería comentar las líneas que llaman a funciones que no necsita, en este caso he comentado las líneas 47, 53, 54, 57 y 66 porque gentoo no las necesita. La única cosas que no debe cambiar son los nombre de las reglas, porque es necesario que se llamen de esta forma para que todas las herramientas que las usan utilizen estos mismos nombres, esto se obliga en nuestro manual de Empaquetamiento.

Por supuesto, se necesita hacer algunos ajustes aquí: en la línea 58 añadiré 'FIXES' porque ese es nombre del fichero de cambios. Para cualquier otra opción por favor lea la página de manual del programa dh_* involucrado.

14.5 Otros ficheros en el directorio debian/.

14.5.1 copyright

Este fichero contiene la información de copyright del paquete. Su formato no está obligado por la Política, pero sus contenidos sí (sección 6.5). Dh_make crea uno por defecto, que tiene este aspecto:

```
This package was debianized by Josip Rodin jrodin@jagor.srce.hr on
Wed, 11 Nov 1998 21:02:14 +0100.

It was downloaded from <fill in ftp site>

Upstream Author(s): <put author(s) name and email here>

Copyright:

<Must follow here>
```

(N. del T.: se traduce el fichero de copyright, dh_make sólo lo ofrece en inglés)

```
Éste paquete fue debianizado por Josip Rodin jrodin@jagor.srce.hr el
Mie, 11 Nov 1998 21:02:14 +0100.

Se descargó de  <rellenar el servidor de ftp>

Autor/es Original/es: <poner el nombre/s de el/los autor/es y
dirección de correo electrónico aquí>
Copyright:

<Debe venir aquí>
```

Las cosas importantes a añidar a este fichero son el lugar donde se obtuvo el paquete y la nota de copyright de éste (debe incluirla entera). Si el copyright es una de las licencias de software libre populares como GNU, BSD o la licencia Artística, puede sólamente referirse al fichero apropiado en /usr/doc/copyright, que existe en todo sistema Debian. Gentoo está licenciado bajo la Licencia Pública General GNU, así que cambiaremos el fichero a esto:

```
This package was debianized by Josip Rodin <jrodin@jagor.srce.hr> on
Wed, 11 Nov 1998 21:02:14 +0100.

It was downloaded from: ftp://ftp.obsession.se/gentoo/

Upstream Author: Emil Brink <emil@obsession.se>
```

```

This software is copyright (c) 1998-99 by Emil Brink, Obsession
Development.

You are free to distribute this software under the terms of
the GNU General Public License.
On Debian GNU/Linux systems, the complete text of the GNU General
Public License can be found in /usr/doc/copyright/GPL file.
```

14.5.2 README.debian (LEEME.debian, n. del t.)

Cualquier detalle extra en discrepancias entre el programa original y su versión debianizada debería estar documentada aquí. Dh_make crea una por defecto, y éste es su aspecto:

```
gentoo for DEBIAN
----------------------

Comments regarding the Package

Josip Rodin <jrodin@jagor.srce.hr>, Wed, 11 Nov 1998 21:02:14 +0100
```

Dado que no tenemos que poner nada aquí – está permitido borrarla. Por cierto, sí, puede renombrar el fichero a README.Debian :–)

14.5.3 changelog

Este es un fichero necesario, tiene un formato especial descrito en el Manual de Empaquetamiento (sección 3.2.3). Este formato es usado por dpkg y otros programas para obtener el número de versión, revisión, distribución y uregncia de su paquete.

Para usted, es también importante, ya que es bueno tener documentados los cambios que haya hecho. También ayudará a la gente que se descarge su paquete para ver si hay algunos temas sin resolver con el paquete que deberían saber. Se salvará como /usr/doc/gentoo/changelog.Debian.gz en el paquete binario.

Dh_make crea uno por defecto, y tiene este aspecto:

```
1  gentoo (0.9.12-1) unstable; urgency=low
2
3   * Initial Release.
4
5  -- Josip Rodin <jrodin@jagor.srce.hr> Wed, 11 Nov 1998 21:02:14 +0100
6
7  Local variables:
8  mode: debian-changelog
9  add-log-mailing-address: "jrodin@jagor.srce.hr"
10 End:
```

La línea 1 es el nombre del paquete, versión, distribución, y urgencia. El nombre debe coincidir con el del paquete fuente, por ahora, la distribución debería ser 'unstable' (inestable, n. del t.) o 'experimental', y la urgencia no debería cambiarse a nada distinto a 'low' (baja, n. del t.). :–>

Las líneas 3 a 5 son una entrada de registro, donde documenta los cambios hechos en esta revisión del paquete (nos los cambios en las fuentes hechas por los autores originales – hay un fichero especial para este propósito, creado por los autores, y que se instala en /usr/doc/gentoo/changelog.gz). Las nuevas líneas deben insertarse justo después de la primera línea que empieza con un asterisco ('*'). Puede hacer esto con dch, emacs (las líneas 7 a 10 son información de modo para el editor Emacs), o cualquier editor de texto que desee. Acabará con algo así:

```
gentoo (0.9.12-1) unstable; urgency=low

 * Initial Release.
 * This is my first Debian package.
```

```
-- Josip Rodin <jrodin@jagor.srce.hr> Wed, 11 Nov 1998 21:02:14 +0100

Local variables:
mode: debian-changelog
 add-log-mailing-address: "jrodin@jagor.srce.hr"
End:
```

Cuando distribuya una nueva versión, debe incrementar el número de versión del paquete. Puede hacer esto con ‘dch –v <versión>–<revisión>‘ y después insertar los comentarios con su editor preferido. Pisa: ¿cómo puede obtener rápidamente la fecha en el formato requerido? Utilize el comando ‘822–date‘ o ‘date –R‘.

Se añade información sobre la nueva versión al principio del fichero ’changelog’. Tendrá este aspecto después:

```
gentoo (0.9.12-2) unstable; urgency=low

 * Comments about the second revision

-- Josip Rodin <jrodin@jagor.srce.hr> Wed, 11 Nov 1998 22:15:39 +0100

gentoo (0.9.12-1) unstable; urgency=low

 * Initial Release.
 * This is my first Debian package.

-- Josip Rodin <jrodin@jagor.srce.hr> Wed, 11 Nov 1998 21:02:14 +0100

Local variables:
mode: debian-changelog
 add-log-mailing-address: "jrodin@jagor.srce.hr"
End:
```

14.5.4 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 arregla este problema marcando los ficheros de configuración de forma que cuando actualizas un paquete no se le pregunta si desea mantener la nueva configuración o no. Lo consigue poniendo la ruta completa a cada fichero de configuración (se encuentran generalmente en /etc) una por línea, en un fichero llamado ’conffiles’.

Gentoo tiene un fichero de configuración, /etc/gentoorc, y meteremos éste en el fichero ’conffiles’. No es necesario tener este último fichero si su programa no tiene ningún fichero de configuración.

14.5.5 dirs

Este fichero especifíca los directorios que se necesitan pero que no crea un proceso de instalación normal (make install). Por defecto, tiene éste aspecto:

```
usr/bin
usr/sbin
```

Observe que la barra precedente no está incluida. Normalmente lo cambiaríamos para gentools a algo así:

```
usr/X11R6/bin
usr/X11R6/man/man1
```

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

14.5.6 postinst, preinst, postrm, prerm

Estos ficheros se llaman scripts de gestión, y debería intentar evitarlos si puede porque son demasiados complejos. Para más información lea el Manual de Empaquetamiento.

14.5.7 manpage.1.ex

Los ficheros que terminan en *.ex son ejemplos de cómo añadir ese tipo de soporte en el paquete. Si usa uno de ellos, editelo y elimine la extensión .ex.

Su programa debería tener una página de manual. Si no la tiene, éste es un esqueleto que puede rellenar. Lea las páginas de manual para `man(7)` para una breve descripción de cómo crear una página de manual. Asegúrese de renombrar este fichero al nombre del programa y modificar la extensión para indicar la sección a la que debería ir. Aquí hay una corta lista:

```
Section |     Description        |     Notes
   1     Comandos de Usuario       Programas ejecutables o scripts.
   2     Llamadas al Sistema       Funciones dadas por el kernel.
   3     Llamadas a Librerías      Funciones dadas por las librerías del sistema.
   4     Ficheros Especiales      Generalmente se encuentra 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 ejecuta el superusuario.
   9     Routinas del Kernel      Llamadas al sistema no estándar.
```

Así que la página de manual de gentoo debería llamarse gentoo.1, o gentoo.1x porque es un programa de X11. Cómo no había página de manual en las fuentes originales la escribí del ejemplo.

14.5.8 menu.ex

Los usuarios de X Windows tendrán un gestor de ventanas con menus que pueden adaptarse para lanzar programas. Si tienen instalado el paquete de menu de Debian, se creará un conjunto de menus para cada programa del sistema para ellos. No se exige en la política de Debian, pero los usuarios seguramente lo apreciarán. Podemos añadir a Gentoo a los menus editando este fichero. Aquí está el fichero que dh_make crea por defecto:

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

El primer campo especifica qué tipo de interfaz necesita el programa (esto es, texto o X11). La siguiente es el menu y submenu dónde debería aparecer. La lista actual de secciones está en /usr/doc/menu/html/ch2.html#s2.2 El tercero es el nombre del programa. El cuarto es el icono para el programa o nada si no hay ninguno. El quinto es el texto que aparecerá en el menu. El sexto es la orden que ejecuta el programa. will appear in the menu. The sixth is the command that runs the program.

Ahora cambiaremos la entrada de menu a ésta:

```
?package(gentoo):needs=X11 section=Apps/Misc \
  title="gentoo" command="/usr/X11R6/bin/gentoo"
```

14.5.9 watch.ex

Puede usar este fichero, junto con los programas `uscan(1)` y `uupdate(1)` (en el paquete devscripts) para vigilar el servidor de donde obtuvo las fuentes originales. Vea las páginas de manuales para más detalles. Gentoo no puede usar esta función así que borraremos el fichero.

Ahora estamos preparados para construir el paquete.

14.6 Pasos Finales

14.6.1 Construir el paquete.

Entre en el directorio principal de Gentoo's (/usr/local/src/gentoo/gentoo–0.9.12/) y ejecute el siguiente comando:

```
dpkg-buildpackage -rfakeroot
```

Esto hará todo por usted, sólo tendrá que dar su clave secreta PGP, dos veces. Una vez haya hecho esto, verá cuatro nuevos ficheros en el directorio /usr/local/gentoo:

- *gentoo_ 0.9.12–1_ i386.deb*

  es el paquete binario completo. Puede usar dpkg o deselect para instalar o eliminar éste como cualquier otro paquete.

- *gentoo_ 0.9.12–1_ i386.changes*

  Mientras sigue trabajando en el paquete, campiará su comportamiento y se le añadirán nuevas funciones. Las personas que descargen su paquete pueden mirar este fichero y ver qué ha cambiado. Este fichero es generado a partir del fichero gentoo–0.9.12/debian/changelog file, y contiene los cambios actuales a la revisión actual del paquete. También lista los ficheros en el paquete. Las largas listas de números son las sumas MD5 para los ficheros. Las personas que descargen estos ficheros pueden probarlos con `md5sum(1)` y si los números no coinciden, sabrán que el fichero está corrupto o ha sido modificado. Éste fichero está firmado con PGP de forma que cualquiera puede estar aún más seguro de que es realmente suyo.

- *gentoo_ 0.9.12.orig.tar.gz*

  Estes es el código fuente original junto de forma que si alguien quiere recrear su paquete desde cero puede hacerlo. O si alguien no está usando el sistema de paquetes de Debian quiere descargarse las fuentes y compilarlo.

- *gentoo_ 0.9.12–1.dsc*

  Este es un sumario de los contenidos del código fuente. Este fichero es generado con el fichero gentoo–0.9.12/debian/control y se usa cuando se descomprime las fuentes con `dpkg-source(1)`. Éste fichero está firmado con PGP de forma que cualquier puede estar seguro de que es realmente suyo.

14.6.2 Comprobar su paquete para encontrar errores.

Ejecute `lintian(1)` sobre su fichero de cambios .changes; éste programa comprobará muchos errores comunes al empaquetar. El comando es:

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

Si parece que hay algunos errores (líneas que comienzan por E:), lea la explicación (líneas N:), corriga errores, y reconstruya con dpkg–buildpackage. Si hay líneas que comienzan con W:, son sólo avisos (warnings, n. del t.), así que puede estar seguro de que su paquete está bien (pero seguramente necesita algún ajuste fino).

Mire dentro del paquete usando un gestor de ficheros como `mc(1)`, o descomprimalo en algún punto temporal usando `dpkg-deb(1)`.

Instale el paquete para probarlo usted mismo. Intente instalarlo en otroas máquinas distintas de la suye y mire bien para detectar errores o avisos.

Más tarde, cuando construya una nueva versión, debería hacer lo siguiente para asegurar la actualización básica del paquete:

- actualizese de la versión previa (y de la versión en la última versión de Debian),
- vuelva a la versión anterior de nuevo,
- instale el paquete como uno nuevo (esto es, sin ninguna verisón instalada previamente),
- desintalelo, reinstalelo y luego purgelo.

14.6.3 Enviando su paquete.

Ahora que ha probado su nuevo paquete en profundidad, necesitará enviar estos ficheros a master.debian.org, usando `dupload(1)`. Primero debe editar el fichero de configuración de dupload. Copie los valores por defecto del /etc a su directorio personal:

```
cp /etc/dupload.conf ~/.dupload.conf
```

Después edite ese fichero (~/.dupload.conf), y encuentre la parte que empieza por '$cfg{master}' y cambie estas líneas (no necesariamente en éste orden):

```
login => getlogin() || $ENV{USER} || $ENV{LOGNAME},
visibleuser => getlogin() || $ENV{USER} || $ENV{LOGNAME},
visiblename => "",
fullname => "",
```

a valores equivalentes a estos (cambien mis valores por los suyos):

```
login => "joy",
visibleuser => "jrodin",
visiblename => "jagor.srce.hr",
fullname => "Josip Rodin",
```

El primero es su login en master.debian.org (es usted ahora un desarrollador oficial de Debian ¿o no? Si no lo es, lea la Referencia del Desarrollador), el segundo y el tercero son partes de su dirección de contacto antes y después de la arroba ('@'), y el cuarto es su nombre completo.

Entonces conecte con su proveedor de Internet, asegurese una vez mas de que está en el directorio /usr/local/src/gentoo, y ejecute la orden:

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

Dupload le preguntará su password en master.debian.org, envíe sus paquetes, y envíe un pequeño anuncio sobre su envío en <debian-devel-changes@lists.debian.org>.

Si vive en Europa, puede usar otras colas de envío en lugar de master. Para más detalles mire lea `dupload(1)`, `dupload(5)` y la Referencia del Desarrollador.

14.6.4 Dónde pedir ayuda.

Antes de que decida preguntar en lugares públicos, por favor simplemente RTFM ("Lea el Jodido Manual", n. del t.). Esto incluye documentación en /usr/doc/dpkg, /usr/doc/debian, /usr/doc/debhelper y las páginas de man/info para todos los programas mencionados en este artículo. Cuando reciva un aviso de fallo (bug report, n. del t.) (sí, avisos de fallos ¡de verdad!) sabrá que es el momento de indagar en el Sistema de Seguimiento de Fallos de Debian (`http://www.debian.org/Bugs/`) y leer la documentación allí.

Si se une a la lista de distribución de Mentores de Debian en <debian-mentors@lists.debian.org> puede unirse con desarrolladores de Debian con experiencia que le ayudarán con las preguntas que pueda tener. Puede subscribirse a ella enviando un correo electrónico a <debian-mentors-request@lists.debian.org> con la palabra 'subscribe' en el tema del mensaje.

Si aún tiene preguntas, hagalas en la lista de distribución de Desarrolladores de Debian en <debian-devel@lists.debian.org>. Puede subscribirse a ella enviando un correo electrónico a <debian-devel-request@lists.debian.org> con la palabra 'subscribe' en el tema del mensaje. Si ya es un desarrollador de Debian debería estar suscrito a ella de todas formas.

Aunque todo funcionara 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 su paquete, y si cometió algún error crítico será bombardeado por correos de centenares de usuarios furiosos de Debian.... Sólo bromeaba :–)

Relájese y prepárse para recibir reportes de fallos, porque hay mucho más trabajo que hacer antes de seguir completamente las políticas de Debian (una vez más lea la *documentación real* para detalles). ¡Buena suerte!