

# Creación de paquetes de Debian

Javier Fernández Sanguino-Peña.  
jfs@computer.org

15 de abril de 1999

## 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.

(Este artículo ha sido publicado por La Espiral y su versión mas reciente se puede encontrar en <http://www.laespiral.org><sup>1</sup>)

## 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 2500 paquetes de software) a veces interesará instalar software que no es parte de la distribución o 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).

---

1

El copyright de este artículo es compartido por su autor y Open Resources. Este artículo se distribuye bajo la licencia de la Revista OR (OR Magazine License). Puedes ver esta licencia en la siguiente URL: <http://www.openresources.com/magazine/license/index.html>.

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.

## 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 “traducirlos” a 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 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, se trata de un simple programa que escribe “Hello world” por la salida estándar.

Se obtienen primero los tres ficheros fuentes del paquete Debian *hello*, 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`.

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 en la figura 1.

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

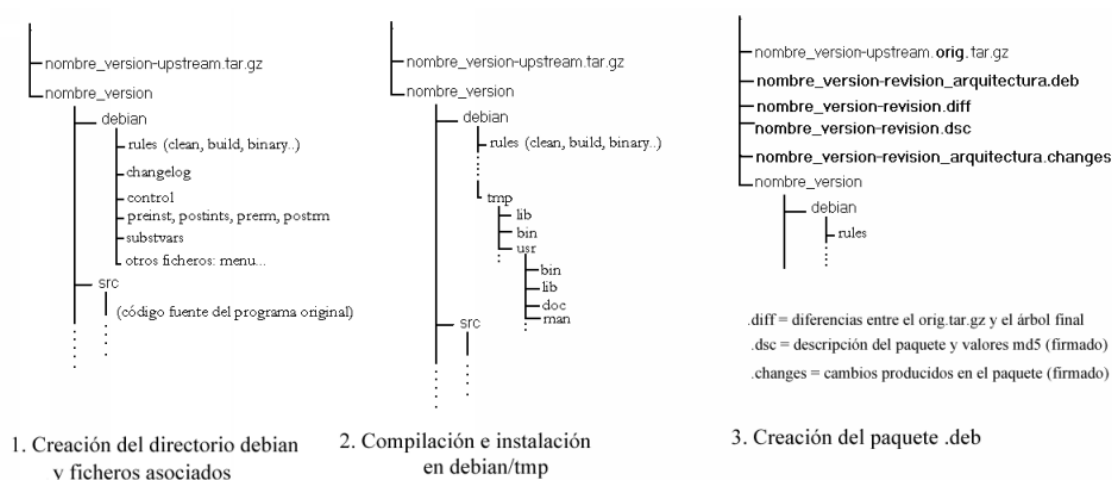


Figura 1: Pasos tomados para construir un paquete

es semi-automático (vía 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.
- **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 desempquetado 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 1) que deben seguir unas reglas definidas en la Política de Paquetes de Debian, 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 asegurarse que la compilación/construcción del paquete se hace sobre una base “limpia”.
- *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 (vía 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 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 sobrescriba 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 actualicen 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 estas es `debmake`, 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* 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 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`, que permite crear paquetes a partir de un repositorio CVS (“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 en-cadenados 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**.
- 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, definiendo un estándar de acceso a la documentación mediante el paquete doc-base (con la documentación en HTML en <http://localhost/doc>) y de instalación de servidor (raíz del servidor y residencia de CGIs) lo que permite a los paquetes integrarse con el servidor local de web.
- su adherencia a los estándares es firme, no sólo existen estándares, sino que se cumplen, existiendo un seguimiento constante de que se cumpla la política definida (ver si no la pagina de Lintian, también disponible como paquete)
- 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.

Y se está trabajando en el uso posible de linuxconf. Habiéndose terminado ya selecciones pre-fabricadas 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
test -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* o *m68k*, pero se está haciendo un gran esfuerzo por llevar a Debian a otras arquitecturas como PowerPC o ARM. 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, modificar 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` (política que se debe seguir para crear paquetes para Debian), y `developers-reference` (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 también son una fuente importante de información, se encuentran indexadas en el servidor de Debian, 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`.

## 13. Acerca de este artículo

La versión original de este artículo está disponible en <http://www.openresources.com/es/magazine/making-debian-packages/>, en la Revista Open Resources.