



Compilación de paquetes

JORGE LÓPEZ
jlopez@iberprensa.com

Los paquetes incluidos en los repositorios de Debian permiten realizar la compilación e instalación de cualquier programa a partir de su código fuente. Este mes en nuestra sección Debian vamos a revisar los pasos necesarios para esta compilación y el uso de apt-build, la herramienta integrada en Debian para simplificar su instalación.

Todo programa está compuesto por uno o más archivos con su código fuente que puede contener desde unas pocas líneas de código a millones de ellas en el lenguaje en el que esté programado. Para poder ejecutar estos archivos, debemos obtener su equivalente en formato binario mediante el proceso de compilación. Este se lleva a cabo con un programa denominado compilador, que *grosso modo* transforma un archivo creado por un programador en otro que la máquina comprende, de nombre *archivo binario*.

En Debian Linux y gracias al uso de una licencia libre, como la GPL, se nos asegura como mínimo la disponibilidad del código fuente de cualquier aplicación acogida a esta licencia, si bien para la mayoría de los programas que nos interesan podemos encontrar también una versión ya compilada.



Parámetros disponibles del compilador GCC.

PAQUETES NECESARIOS

Para crear el ejecutable binario de un programa a partir de su código fuente, como mínimo es necesario un compilador adecuado para el lenguaje en el que esté desa-

rollado, disponer de las librerías de las que dependa y la utilidad *make*.

Las librerías son archivos que contienen funciones utilizadas por el programa para funcionar, por lo que son requeridas para compilarlo.

La herramienta *make* es la encargada de realizar el proceso de compilación, para lo cual lee el archivo *Makefile*, que contiene un listado con los ficheros y directorios necesarios para la compilación.

Para obtener los paquetes básicos, casi siempre con el código en lenguaje C, necesarios para realizar la mayoría de compilaciones, ejecutamos:

```
# aptitude install
build-essential
```

que descarga los compiladores GCC y G++, para los lenguajes C y C++ respectivamente, las librerías *libc6* y la herramienta *make*.

Cada programa puede requerir unas determinadas dependencias por lo que será necesario disponer de las librerías que haga uso para ejecutarse. Estas librerías muestran el sufijo *-dev* en los repositorios de Debian, por ejemplo, *libsvm-dev*.

Las librerías las podemos obtener de los repositorios de Debian o desde el paquete disponible en su página oficial.

En caso de que el paquete que queremos compilar se encuentre solo en su formato binario en los repositorios de Debian, ejecutando:



Actualizando las librerías existentes en el equipo.

```
# apt-get build-dep paquete
```

se descargarán e instalarán las librerías y archivos necesarios para poder realizar su compilación.

Como último comando necesario para terminar los requisitos de una compilación, si instalamos una librería directamente desde su paquete original, sin utilizar *aptitude*, es necesario ejecutar:

```
# ldconfig
```

que actualiza el listado de librerías disponibles.

PASOS COMUNES PARA LA COMPILACIÓN

El proceso de compilación de la mayoría de paquetes se reduce a una serie de pasos comunes que podemos realizar sin necesidad de disponer de conocimientos avanzados y que pasamos a describir:

PASO 1

Los paquetes con el código fuente se encuentran habitualmente en formato comprimido, TAR.GZ y TAR.BZ2. Por lo tanto, una vez descargados a nuestra máquina, en un directorio temporal, por ejemplo */tmp*, es necesario descomprimirlos ejecutando:

```
$ tar xvzf paquete.tar.gz
```

para los TAR.GZ, o:

```
$ tar xvjf paquete.tar.bz2
```

para los TAR.BZ2.

Al realizar esta descompresión se crea un nuevo directorio con el mismo nombre que el paquete inicial. Este directorio contiene los archivos necesarios para compilar el programa y obtener el ejecutable binario.

Un paquete compilado a partir del código fuente, aumenta de un 5% a un 20% el rendimiento del equivalente binario

copiamos a los directorios del sistema los archivos que componen el programa. Esta ruta destino es para la mayoría de paquetes `/usr/local`.

Al utilizar este comando podríamos iniciar el programa sin tener en cuenta el directorio en el que estemos situados con:

```
$ ejecutable
```

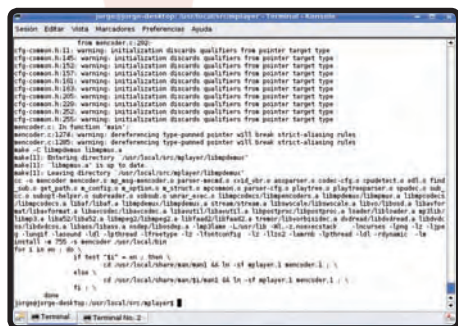
Para eliminar los archivos copiados con el comando anterior, ejecutamos:

```
# make uninstall
```

Un último comando muy utilizado de `make` es

```
$ make clean
```

que limpia todos los archivos creados en el proceso de compilación, lo cual nos permite realizar el proceso de nuevo desde cero.



Listado de archivos copiados con el parámetro "install".

EXCEPCIONES EN LA ECOMPILACIÓN

La mayoría de paquetes se compilan siguiendo los pasos que acabamos de describir, pero aquellos con un tamaño muy pequeño, los destinados a más de un sistema operativo o los desarrollados con otro lenguaje de programación que no sea C, pueden presentar excepciones que necesitan otros métodos de compilación:

Existen varios archivos Makefile, cada uno para un sistema operativo distinto

Algunos paquetes, como el emulador `xmame`, presentan más de un archivo `makefile`, destinados cada uno a una plataforma concreta. En esta situación, debemos indicar cuál queremos compilar. Por ejemplo, en `xmame`, elegiríamos el archivo para Unix, de nombre `makefile.unix`, ejecutando:

```
$ make -f makefile.unix
```

donde se utiliza el parámetro "-f" para especificar el archivo `Makefile` concreto.

Solo está disponible un archivo con extensión C o CPP, sin archivo configure

Los ficheros binarios de estos archivos deben crearse directamente desde su código fuente con el compilador `gcc` para la extensión C o `g++` para CPP.

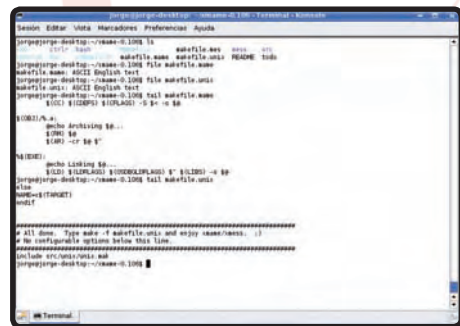
Por ejemplo, si tenemos el fichero `archivo.c`, creamos su binario ejecutando:

```
$ gcc archivo.c -o archivo
```

donde con el parámetro "-o" indicamos el nombre del ejecutable resultante.

Otras extensiones de archivos

Aunque la mayoría de paquetes están desarrollados en C o C++, algunos utilizan otros lenguajes de programación como Java o Pascal. Para los fuentes que utilicen estos lenguajes es necesario emplear otras herramientas, principalmente los compiladores `java` y `gpc`. El equivalente en Java a `make`, es `ant`, cuya configuración se basa en archivos XML.



Varios archivos make de un mismo programa.

EJEMPLO CON MPLAYER Y MENCODER

Para comprender el proceso de instalación de una aplicación compleja a partir de su código fuente, vamos a compilar el reproductor MPlayer y el codificador `mencoder`, disponibles en el mismo paquete, con soporte para la decodificación en MP3. Ambos programas poseen varias librerías como dependencias, que son necesarias para que se puedan leer el máximo número de formatos de archivos multimedia.

Para realizar su compilación seguimos los siguientes pasos:

PASO 1

Primero instalamos el paquete `lame`, que proporciona las librerías del formato MP3 para `mencoder`.

Descargamos el paquete con su código fuente:

```
$ wget http://easynews.dl.sourceforge.net/sourceforge/lame/lame-3.97.tar.gz
```

lo descomprimos:

```
$ tar xvzf lame-3.97.tar.gz
```

accedemos al nuevo directorio:

```
$ cd lame-3.97
```

creamos el archivo `Makefile`:

```
$ ./configure
```

y compilamos e instalamos:

```
$ make; make install
```

invocando en una misma línea la ejecución primero `make` y después `make install`, mediante el caracter " ; ".

PASO 2

Al ejecutar:

```
$ ffmpeg
```

comprobamos que se produce un error en la carga de una librería. Esta situación se ha producido por no encontrarse en el directorio `/usr/lib` las librerías necesarias, que se encuentran en `/usr/local/bin`, directorio que no está accesible para `ffmpeg` en su ejecución.

Para solucionarlo, creamos un enlace simbólico en `/usr/lib` de todas las librerías para las que se indique error al ejecutar `ffmpeg` hasta que se muestren correctamente las opciones disponibles para el programa. La creación de estos enlaces simbólicos la realizamos mediante la herramienta `ln` junto al parámetro "-s", por ejemplo:

```
# ln -s /usr/local/lib/libavformat.so.52 /usr/lib/libavformat.so.52
```

donde indicamos primero su directorio origen y después su ruta destino.

PASO 3

A continuación, procedemos a descargar la última versión del paquete MPlayer utilizando el cliente de subversion:

```
$ svn checkout
svn://svn.mplayerhq.hu/mplayer
/trunk mplayer
```

creándose en la ruta actual un nuevo directorio de nombre *mplayer*.

► PASO 4

Como hemos hecho para *ffmpeg*, realizamos su compilación con los siguientes comandos:

```
$ ./configure
$ make
```

pudiéndose ejecutar ambos programas desde el directorio actual:

```
$ ./mplayer
$ ./mencoder
```

o copiarlos a los directorios correspondientes del sistema:

```
# make install
```

PROBLEMAS MÁS COMUNES AL COMPILAR

La mayoría de los paquetes que compilamos no nos deberían producir problemas, y seguramente podremos seguir todos los pasos satisfactoriamente.

Pero en ocasiones, debido a la utilización de librerías concretas, que dispongamos de una versión obsoleta de las mismas, de utilizar un compilador distinto, o no disponer de los permisos adecuados, es posible encontramos con algunos problemas comunes que en principio son fáciles de resolver. Podríamos resumir los más habituales en los siguientes:

► Error en la ejecución de *make install*

El directorio utilizado para la instalación de los archivos que componen un programa, habitualmente */usr/local*, solo cuenta con los permisos de escritura para root. Por lo tanto, debemos utilizar este usuario para la instalación.

► No está disponible el ejecutable del programa compilado

Según la configuración del sistema y el directorio en el que se haya instalado el programa, podremos iniciar el ejecutable del programa desde cualquier directorio o solo indicando su ruta exacta. En caso de que no podamos lanzarlo utilizando solo su nombre, deberemos indicar su directorio completo, por ejemplo:

```
$ /usr/local/bin/ejecutable
```

► La creación del fichero *Makefile* produce un error distinto la falta de un archivo.

Este error suele ocurrir cuando las versiones de las librerías instaladas son más antiguas de las requeridas por el programa, por tanto tendremos que actualizarlas. En caso de persistir el problema, podemos ampliar la información del error en el archivo *config.log* que se genera automáticamente, donde justo a su final se muestra el punto exacto donde se ha producido y su causa.

HERRAMIENTA APT-BUILD

Para simplificar la compilación de nuevos paquetes, Debian incluye la herramienta *apt-build*. Esta se encarga de instalar los paquetes necesarios para compilar, descargar sus dependencias y el código fuente del programa, compilarlo e instalarlo.

Para poder utilizar *apt-build*, debemos incluir un nuevo repositorio para ser utilizado por *apt*; por lo que abrimos el archivo */etc/apt/sources.list* y añadimos la línea:

```
deb-src http://http.us.debian.org/
debian stable main contrib
non-free
```

sustituyendo *stable* por la versión que poseamos, en caso de que no utilicemos esta.

A continuación actualizamos el listado de paquetes disponibles:

```
# aptitude update
```

e instalamos el paquete *apt-build*:

```
# aptitude install apt-build
```

Al descargar este paquete, se abre la herramienta *debconf* para que seleccionemos el nivel de optimización que va a utilizarse al compilar. De los tres disponibles, el aconsejado es el nivel medio, correspondiente al parámetro “-O2”, ya que el alto puede producir errores aleatorios en la posterior ejecución de los programas.

Aceptamos que se añada el nuevo repositorio, destinado a ser utilizado como origen de las compilaciones.

Para dar por finalizada la instalación, elegimos la arquitectura del equipo en el que se va a instalar *apt-build*, obteniendo las optimizaciones adecuadas para este.

Ya podemos realizar la descarga y compilación de los nuevos paquetes disponibles en el repositorio *deb-src* con solo ejecutar:

```
# apt-build install paquete
```

Por ejemplo, para compilar el gestor de archivos MC, ejecutamos:

```
# apt-build install mc
```

En caso de indicar un paquete ya instalado en formato binario, se va a proceder igual a la compilación del código fuente de este, actualizándose como si se tratara de un nuevo paquete. ■

En el próximo número ►

Seguiremos en la línea de este tutorial y estudiaremos la creación de paquetes DEB, tanto a partir de otros formatos binarios, como desde su código fuente. También revisaremos su redistribución a raíz de un repositorio propio en una red o Internet.

Actualidad Debian ►

► Nuevos paquetes

Los últimos paquetes añadidos a los repositorios de Debian han sido: *boswars*, juego de estrategia en tiempo real ambientado en un mundo futurista; *emdebian-archive-keyring*, conjunto de claves del repositorio *emdebian*, destinado a ser utilizado en sistemas embebidos; *flog*, lee la salida de STDIN y la escribe en archivos, permitiendo generar logs fácilmente; *g15stats*, monitor de carga de CPU, memoria y swap, orientado a teclados Logitech que incluyen pantallas LCD; *josm*, editor libre para el sistema geográfico OpenStreetMap; *jscoverage*, generador de estadísticas para programas JavaScript; *mauve*, testea el contenido de librerías Java; *pathfinderd*, demonio de detección y validación de certificados X.509; *php5-radius*, módulo de autenticación de Radius para PHP; *phpgroupware*, herramienta de trabajo en grupo, incluye calendario, webmail, editor de texto y agenda de contactos; *python-elisa*, media center basado en GStreamer; *qtnx*, cliente de conexión NX programado con QT; *simutrans*, juego de simulación de transportes; y *tangerine*, servidor destinado a compartir música, con soporte del protocolo DAAP.