Creación de paquetes .deb

Un simple *script* que no requiere más que un simple "ejecutar", se distribuye muy fácilmente. Pero ¿qué sucede con las aplicaciones con gran cantidad de archivos? Sobretodo, con aquellas aplicaciones donde un *tarball* no simplifica la tarea de repartir los ficheros que deban almacenarse en rutas diversas. Sin dudas, saber crear un paquete Debian, es la forma más segura de crear un instalador para nuestra aplicación.

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y **eXtreme Programming**. Miembro de la **Free Software Foundation**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim. Fundadora y Responsable Editorial de **Hackers & Developers Magazine**.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:
Twitter / Identi.ca: **@eugeniabahit**

En ediciones anteriores de Hackers & Developers Magazine, comentábamos acerca de cómo distribuir aplicaciones y/o módulos Python a través del Python Package Index para luego instalarlos mediante un simple `pip install`. Incluso, más adelante, estuvimos viendo cómo se crean las páginas man del Manual de GNU/Linux. Para continuar con esta serie de procesos básicos que todo programador experto debe conocer, hoy, en unos pasos muy sencillos, tomaremos nota de cómo se crean los archivos `.deb`

Preparando la estructura de directorios

Lo primero que se debe tener bien en claro, es cuál será la ruta o rutas de destino de la aplicación al momento de desempaquetar el .deb resultante. Teniendo en claro esto, se debe crear la estructura de directorios correspondiente. ¿Cómo? Primero necesitamos una carpeta donde trabajar:

```
mkdir myproject
```

Dentro de esa carpeta, es donde debemos emular (o clonar) las rutas de destino de la aplicación. Por ejemplo, imaginemos una aplicación Web que debe colocarse en el directorio `/var/www/myapp` pero que además, requiere que en e directorio `/var/log` se cree un directorio `myapp`:

```
mkdir -p myproject/var/www/myapp
mkdir -p myproject/var/log/myapp
```

De esta forma, la estructura de directorios se vería como la siguiente:

```
myproject/
└── var
    ├── log
    │   └── myapp
    └── www
        └── myapp
```

El siguiente paso, será copiar dentro de las carpetas creadas, todos los archivos que sean necesarios. Es decir que, si nuestra aplicación se encuentra en una carpeta llamada MyApp1_2_0, todo el contenido de esa carpeta, debería copiarse dentro `myproject/var/www/myapp`.

Generando un archivo de control del paquete Debian

Todo paquete Debian (archivo .deb) debe contener un directorio **DEBIAN** con sus archivos correspondientes. Lo primero entonces, será crear la carpeta DEBIAN dentro de nuestro directorio de trabajo:

```
mkdir myproject/DEBIAN
```

Existen **19 tipos de archivos** entre los que se "deben" colocar y los que se "pueden". Esto significa que mientras algunos archivos serán obligatorios, otros, serán opcionales a tal punto que en muchos casos, si no son estrictamente necesarios, se "debe" evitar colocar dichos archivos en el paquete.

La lista de archivos obligatorios puede obtenerse junto a su detalle correspondiente, ingresando en http://www.debian.org/doc/manuals/maint-guide/dreq.es.html mientras que la de los opcionales, se encuentra en http://www.debian.org/doc/manuals/maint-guide/dother.es.html.

Entre los archivos que se "deben" colocar, se encuentra el archivo **`control`** quien contiene ciertas directivas que serán utilizadas por gestores de paquetes como `apt`, `aptitude` y `dpkg` -entre otros-, al momento de instalar (o mejor dicho, "desempaquetar") el .deb.

Las directivas completas del archivo control, se encuentran descritas en el **Capítulo 5** del **Manual de Políticas de Debian**, el cual puede ser consultado desde http://www.debian.org/doc/debian-policy/ch-controlfields.html.

El archivo control se encuentra **dividido en dos bloques**: el primero, contiene información pura y exclusivamente sobre los fuentes del paquete (aquellos archivos requeridos para armar y construir la aplicación final que utilizará el usuario), mientras que el segundo, la contiene sobre el binario de la aplicación en sí misma (es decir, la aplicación que finalmente será utilizada).

De las **directivas del archivo DEBIAN/control**, las más interesantes (y casi ineludibles), son las siguientes:

Package
Nombre del binario. Sigue las mismas reglas de nombre que la directiva source del bloque 1.

Priority
Indica la importancia que la instalación del paquete implica para el usuario. Los diferentes niveles de prioridades de un paquete Debian, pueden ser consultado en http://www.debian.org/doc/debian-policy/ch-archive.html#s-priorities. No obstante, los mismos no necesitan ser demasiado explicados ya que poseen nombres sumamente descriptivos:

```
required, important, standard, optional, extra.
```

Section
Área de aplicación en la cual se ha clasificado el paquete. Las secciones pueden ser:

```
admin, cli-mono, comm, database, debug, devel, doc, editors, education, electronics, embedded,
fonts, games, gnome, gnu-r, gnustep, graphics, hamradio, haskell, httpd, interpreters,
introspection, java, kde, kernel, libdevel, libs, lisp, localization, mail, math,
metapackages, misc, net, news, ocaml, oldlibs, otherosfs, perl, php, python, ruby, science,
shells, sound, tasks, tex, text, utils, vcs, video, web, x11, xfce, zope
```

Maintainer
Nombre y correo electrónico del desarrollador original. Debe guardar el formato:

```
Nombre Apellido <usuario@dominio.foo>
```

Version
Versión del paquete binario (para entender mejor como versionar de manera estándar una aplicación, te recomiendo leer el Anexo I al final de este artículo).

Architecture
Generalmente suele utilizarse como valor **any**, cuando el binario se ha escrito en lenguaje compilado y **all**, si se tratase de un lenguaje interpretado. No obstante, una referencia completa puede obtenerse en http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Architecture.

Description
Descripción de la aplicación, dividida en dos sectores: descripción corta (primera línea) y descripción larga (líneas siguientes). Los dos sectores de descripción, deben contener el siguiente formato:

```
descripción corta de hasta 60 caracteres
  descripción larga, tabulada con espacios en blanco
  que además, puede ser de varias líneas.
```

Depends, Recommends, Suggests
Muy descriptivos los nombres de estas directivas, indicarán de cuáles paquetes depende la aplicación y cuáles son recomendados y sugeridos para el mejor funcionamiento de ésta.

Homepage
La URL del paquete y preferentemente se recomienda colocar aquella de la cuál el paquete pueda ser descargado.

Un **ejemplo** completo del archivo **DEBIAN/control**, podría verse de la siguiente manera:

```
Package: europiocli
```

```
Priority: extra
Section: php
Maintainer: Eugenia Bahit <ebahit@member.fsf.org>
Version: 3.2.17
Architecture: all
Description: Command Line Interface para Europio Engine
  Interfaz de línea de comandos que permite agilizar tareas de comunes
  en Europio Engine, como crear o eliminar módulos; agregar modelos,
  vistas y controladores, etc.
Depends: php5 (>=5.3.10), php5-cli (>=5.3.10), mysql-server (>=5.5.0)
Homepage: http://europio.org/some/dir/to/download/current-package
```

> Por favor, nótese que las dependencias son separadas por una coma seguida de un espacio en blanco.

Entonces, creamos el archivo control con el contenido anterior, reemplazando el valor de cada directiva según corresponda:

```
touch myproject/DEBIAN/control
```

Construyendo el .deb

Para construir el paquete .deb se utilizará `dpkg`. Para evitar el uso de `fakeroot` al momento de construir el paquete con `dpkg` es necesario modificar el propietario de los archivos de forma recursiva:

```
sudo chown -R root:root myproject/
```

Si **dpkg-dev** no se encuentra instalado, debe instalarse ya que será necesario para la construcción:

```
 sudo apt-get install dpkg-dev
```

Para construir el paquete, solo habrá que situarse en el nivel superior del directorio de trabajo y ejecutar:

```
dpkg -b myproject destino.deb
```

Se puede probar el paquete utilizando la opción -i de dpkg:

```
dpkg -i mipaquete.deb
```

Anexo I: Estándar para asignar números de versiones

El número de versión de un Software se encuentra conformado por cuatro componentes:

- Mayor y menor número de versión
- Mayor y menor número de revisión

Mayor número de versión

El mayor número de versión, debe ser modificado cuando el cambio hecho a la aplicación, requiere cambiar todo el paquete.

Menor número de versión

Éste, debe ser modificado cuando se realizan cambios significativos en la aplicación que requieran modificar una parte importante del paquete pero no todo.

Mayor número de revisión

Ante cualquier cambio de sentido y/o significado (por más mínimo que sea) de la aplicación, el número de revisión mayor es modificado.

Menor número de revisión

Se modifica ante cambios menores que generalmente tienen que ver con pequeñas cuestiones estéticas o correcciones de errores de mecanografiado, pero que no implican un cambio de sentido en la aplicación.

> Solo los 3 primeros componentes del número de versión son los que importan y tienen un verdadero significado. Frente a números de versión como 2.1.0.0 se prefiere utilizar 2.1.0 directamente.

En entregas posteriores, intentaremos abarcar el estudio de otros archivos del directorio DEBIAN, procurando ahondar además, en la ejecución de *Scripts* post-instalación.