Anatomía de un Slackbuild

Preámbulo

Supongo que todos los usuarios de Slackware habrán usado alguna vez un script de SlackBuild para crear un paquete de software que pueda ser instalado fácil y limpiamente y eliminado más tarde si es necesario. Mi experiencia fue que ejecuté el script de SlackBuild para crear un paquete y nunca miré realmente el código que contenía hasta que un día la versión fuente no coincidió con la indicada en el script.

¿Por qué te molestarías en mirar el código en un script de SlackBuild? Bueno, en primer lugar, puedes usarlo si estás aprendiendo a hacer un bash. En segundo lugar, puede que no quieras enviar un script de SlackBuild, pero quizás quieras crear un paquete que no exista para el software que quieres.

Así que aquí me gustaría revisar un script de SlackBuild lo mejor que pueda (verbalmente). Doy la bienvenida abiertamente a las ediciones de todos los demás en todos los niveles. De hecho, creo que es importante que la gente de todos los niveles contribuya. Lo que es obvio para una persona puede necesitar ser explicado a otra persona.

Existen diferentes enfoques de un script de Slackbuilds, por ejemplo Alien Bob tiene su muy útil creador de SlackBuild en: Herramienta de Alien para un SlackBuild.

En estos días en SlackBuilds.org prefieren las presentaciones usando una plantilla de SlackBuild. Creo que sería bueno utilizar un script de SlackBuild que se encuentra actualmente en SlackBuilds.org. Para ello utilizaré el script de SlackBuild que envié para latex2html a Slackbuilds.org latex2html y que esta actualmente disponible para la versión de Slackware 14.2.

De esta manera, estamos hablando acerca de un script contemporáneo que se puede descargar, usar y con suerte relacionarte con el una vez que leas esto.

La otra cosa que hay que mencionar es que el **contexto** de esta página, cuando lo lees, con respecto a un script latex2html es que se ha descargado todo un “slackbuild” de slackbuilds.org, está en algún lugar conveniente (digamos en mi escritorio) que se desempaqueta y en el directorio desempaquetado llamado “latex2html” se ha colocado el código fuente del software llamado latex2html-2019.2.tar.gz.

Este script se ejecuta rápidamente de la siguiente forma:

```
bash-5.0# chmod a+x latex2html.SlackBuild
bash-5.0# ./latex2html.SlackBuild
```

Bash

En los días anteriores a Windows en Unix, un shell o Bourne Shell (Stephen Bourne, Bell Labs) fue una forma de comunicarse con el sistema. En reconocimiento a Stephen Bourne, Brian Fox lanzó una nueva versión en 1989 y la llamó *B**ourne** a**gain** Sh**ell (bash).

Si abrís una terminal en Slackware - es decir `Menu → System → Console` y tipeando en el $

prompt:

```
bash --version
```

, deberías conseguir algo como:

```
GNU bash, version 5.0.11(1)-release (x86_64-slackware-linux-gnu)
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
```

En términos simples, la entrada fue recibida y actuada. La entrada tiene una duración algo corta, ya que cuando cierras la ventana la entrada ha desaparecido.

Un script en bash en términos simples es un código en bash que puede ser salvado como un archivo en un disco duro.

La primer línea de un script en bash tiene la línea:

```
#!/bin/sh
```

Esto es comúnmente referido como un *shebang*. Esto define que el archivo es un shell script y también el camino al interprete shell.

Script SlackBuild

Hasta ahora hemos descrito que un script de SlackBuild es en esencia código bash presentado en una presentación más permanente de un archivo.

En la parte superior del archivo esta el *shebang*.

Después de esto, en línea con los requisitos de slackbuilds.org, las siguientes líneas son el nombre del paquete, quién escribió el script y opcionalmente se añade un aviso de copyright.

Intentaré revisar el 14.2 latex2html de slackbuild línea por línea. Si hay espacios en blanco, es un aviso para que otros contribuyan. Probablemente también significa que no entiendo completamente esa parte.

```
PRGNAM=latex2html
VERSION=${VERSION:-2019.2}
BUILD=${BUILD:-1}
TAG=${TAG:-_SBo}
```

Si miro el paquete que he instalado en mi actual Slackware de 64 bits es: `latex2html-2019.2-x86_64-1_SBo`. Ahora si miras la primera línea de arriba verás PRGNAM (nombre del programa). Esta es una variable y su valor se establece en la cadena `latex2html`.

La versión esta relacionada a la fuente del software; si vas a la pagina web fuentes de Latex, podrás

observar que el código fuente es del 5 de junio de 2019. Por lo tanto, simplemente le he puesto un nombre a la versión después de esa versión. Para aquellos que están aprendiendo bash (yo todavía lo estoy): en la segunda línea VERSION= de la izquierda asigna un valor a la variable llamada VERSION.

Cuando se asigna un valor a la variable VERSION, ¿por qué usamos VERSION=${VERSION:-2019.2} y no simplemente VERSION=2019.2?
Tome la expresión ${VERSION}. Esto significa “el valor contenido en la variable VERSION”. La notación más compleja ${VERSION:-2019.2} significa “el valor contenido en la variable VERSION, pero si esa variable no tiene todavía un valor entonces usa el valor por defecto de '2019.2'”.
Así que esa segunda línea simplemente se reduce a VERSION=2019.2. Se puede establecer un valor para VERSION fuera del script: si especificaste un valor para VERSION en la línea de comandos de SlackBuild o si se ha definido en tu entorno Bash.

De la misma forma la variable TAG es configurada a SBo y cuando el paquete es creado esto muestra esto es un paquete “slackbuilt”. Si miras en los repositorio de los paquetes, deberías ver en el nombre del paquete que es de Alien Bob, por ejemplo chromium-77.0.3865.75-x86_64-1alien.

El próximo bloque de código es como sigue:

```
if [ -z "$ARCH" ]; then
  case "$( uname -m )" in
    i?86) ARCH=i586 ;;
    arm*) ARCH=arm ;;
    *) ARCH=$( uname -m ) ;;
  esac
fi
```

En un lenguaje de programación tales como php, python y otros incluyendo bash hay algunas practicas comunes y lógicas. Un principio común es el procedimiento de tener una prueba de código. Una evaluación es aplicada y por supuesto que habrá una salida dependiendo del resultado. Un ejemplo de una sentencia “if” en un script bash es:

```
if[ condición a evaluar]
then
código a ser ejecutado dependiendo del resultado
fi
```

En el bloque anterior, simplemente el “if” es el comienzo de una condición de prueba If y “fi” denota el final de una condición de prueba If.

Volviendo al bash por un minuto, se puede establecer una variable bash de la siguiente manera

```
ARCH=something
```

y podes obtener el valor de una variables y ver el valor de la variable “ARCH” colocando un signo dolar de la siguiente forma:

```
echo $ARCH
```

Una bandera (flag) -z puede ser utilizada en una sentencia “if” para ver si una cadena de texto esta

vaciá. Así que echemos un vistazo a la primera línea de nuevo y averigüemos lo que significa.

```
if [ -z "$ARCH" ]; then
```

En lo anterior no necesitamos ver el valor de ARCH; siempre y cuando podamos acceder a su valor y ser capaces de usarlo en el script. Así que la primera parte antes del “then” simplemente equivale a, si el valor de la variable ARCH se expresa como una cadena y su valor está vacío, haz algo. ARCH podría representar un valor de cadena de la arquitectura del PC en el que se está ejecutando el script.

Otro algoritmo común es llamado, por ejemplo, php es la “sentencia switch”. En términos simples, es una lista con condiciones que se va evaluando. Si ninguna de las condiciones se cumple en las últimas líneas puede colocar lo que le gustaría hacer.

En bash en principio es la misma idea, pero se le llama “sentencia case”. Podes colocar lo que desees en los bloques case, pero si piensas en el hecho de que queremos averiguar la “arquitectura” de un ordenador y ya sabemos que sólo hay ciertas posibilidades, entonces tiene sentido probar primero un par de posibilidades conocidas en la lista y, si no hay ninguna coincidencia, hacer algo para obtener una respuesta. arm y i586 son dos tipos de arquitecturas de computadoras.

Ahora, si pruebas este código en una ventana de terminal:

```
uname -m
```

Esto muestra la arquitectura de tu PC, en mi caso es x86_64.

Así que para resumir con respecto al bloque de código. Primero una sentencia “if” se ejecuta para ver si la variable “ARCH” está vacía. Si hay un valor para la variable ARCH, nada en el bloque de “if” y “case” interno se ejecutará; pero si una cadena vacía es encontrada (ARCH no tiene valor) una “sentencia case” es ejecutada (dentro del bloque de código if) para buscar una coincidencia. Si se encuentra una coincidencia la variable ARCH sera asignada con el valor de la coincidencia y la ejecución del case se detendría. Si la variable ARCH estaba vacía y no coincide con la lista, entonces entra en juego la línea que dice:

```
uname -m
```

es usado para obtener el resultado y la variable ARCH es fijada con el resultado.

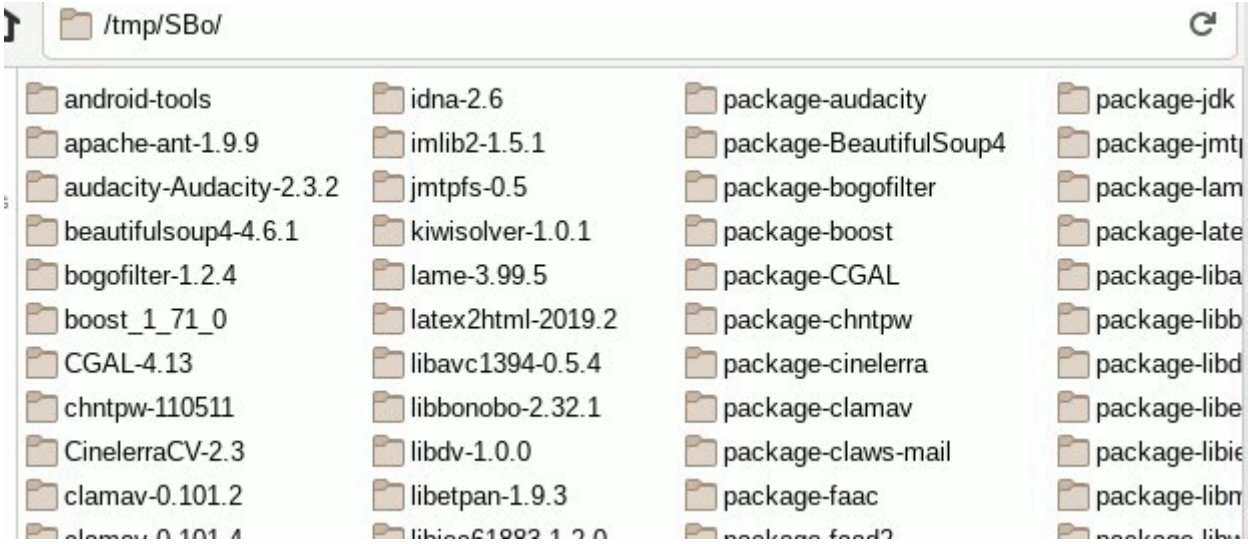
No te preocupes por el signo de interrogación en i?86, el signo de interrogación es un lugar que permite posibilidades a través de regex. Podría ser “3” (i386), “6”(i686), etc.

Próximo bloque de código

```
CWD=$(pwd)
TMP=${TMP:-/tmp/SBo}
PKG=$TMP/package-$PRGNAM
OUTPUT=${OUTPUT:-/tmp}
```

Antes de entrar en esto, déjame echar un vistazo a mi sistema de archivos de Slackware y ver qué hay en /tmp/SBo. Echando un vistazo rápido a la imagen te dará una pista de que el slackbuild funciona, usando el directorio /tmp/SBo/ y crea un directorio con la sintaxis package-packagename. Así que si ahora echamos un vistazo al código anterior. CWD (directorio de trabajo actual) es una

variable y es configurada al valor de pwd. Si lo ejecutas en una ventana del terminal, te dirá dónde estás en un contexto bash desde el que estás trabajando.



TMP se va a establecer en /tmp/SBo. Ahora echemos un vistazo a

```
PKG=$TMP/package-$PRGNAM
```

Se podría suponer que la PKG se va a fijar para el látex2html a:

```
/tmp/SBo/package-latex2html
```

Si se mira de cerca la imagen (tomando en cuenta /tmp/SBo/ en la parte superior de la imagen) verás exactamente que en la imagen. OUTPUT está ajustado a /tmp

Próximo bloque de código:

```
if [ "$ARCH" = "i586" ]; then
  SLKCFLAGS="-O2 -march=i586 -mtune=i686"
  LIBDIRSUFFIX=""
elif [ "$ARCH" = "i686" ]; then
  SLKCFLAGS="-O2 -march=i686 -mtune=i686"
  LIBDIRSUFFIX=""
elif [ "$ARCH" = "x86_64" ]; then
  SLKCFLAGS="-O2 -fPIC"
  LIBDIRSUFFIX="64"
else
  SLKCFLAGS="-O2"
  LIBDIRSUFFIX=""
fi
```

Probablemente, antes de mirar el resto del código del slackbuild latex2html para introducir necesitemos algunos conceptos básicos.

Históricamente el software para computadoras es instalado en un un proceso de tres pasos llamado configure, make and make install. configure se utiliza para preparar la construcción del software, comprueba que todo lo necesario está en el sistema y crea un archivo para make. Un archivo para

make es un archivo que contiene instrucciones para compilar un programa.

A partir de la línea de comandos se puede instalar software solamente usando configure, make and make install. Un slackbuild hace el trabajo de crear el paquete para que el proceso de instalación sea más manejable y confiable de moda. Cuando usted tiene un slackbuild descargado en su sistema, si hay una nueva versión del código fuente es una simple cuestión de poner esa fuente en su slackbuild desempaquetado y una rápida edición del script de slackbuild.

El código fuente de los programas informáticos está escrito en lenguajes de “alto nivel”, pero se convierte en una forma que el ordenador puede entender fácilmente. El código escrito en el lenguaje C implica que un compilador de C lo convierte a binario.

El objetivo de cualquier sistema que instale un programa, es que debe implicar el concepto de hacerlo “a la medida” del ordenador que se está instalando. Obviamente eso va a implicar la arquitectura del ordenador. Durante el proceso de compilación el sistema puede ser ajustado pasando opciones de variables.

Así que ahora echemos un vistazo al bloque de código de arriba. El bloque de código es simplemente un “bloque de if, else”, donde el código se ejecuta de arriba a abajo y asciende a -si la arquitectura es i586 pongan SLKFLAGS a .. si no vayan a la siguiente línea.

CFLAGS y CXXFLAGS son variables que contienen valores que pueden ser pasados en tiempo de compilación. Veremos más tarde que la variable SLKFLAGS se utilizará para establecerlas.

Próximo bloque de código:

```
set -e
rm -rf $PKG
mkdir -p $TMP $PKG $OUTPUT
cd $TMP
rm -rf $PRGNAM-$VERSION
tar xvf $CWD/$PRGNAM-$VERSION.tar.gz
cd $PRGNAM-$VERSION
chown -R root:root .
find -L . \
 \( -perm 777 -o -perm 775 -o -perm 750 -o -perm 711 -o -perm 555 \
  -o -perm 511 \) -exec chmod 755 {} \; -o \
 \( -perm 666 -o -perm 664 -o -perm 640 -o -perm 600 -o -perm 444 \
  -o -perm 440 -o -perm 400 \) -exec chmod 644 {} \;
```

set -e: esto detiene la ejecución del script si hay un error al ejecutar este código siguiendo este comando
rm -rf $PKG: esto borra algún directorio previo (y contenido) de rm -rf $PKG: this deletes any previous directory (and contents) of package-latex2html en /tmp/SBo/package-latex2html

package-latex2html at of /tmp/SBo/package-latex2html
mkdir -p $TMP $PKG $OUTPUT :mkdir con la bandera (flag) “ -p ” crea directorios, pero sólo si no existen ya.

que serían /tmp/SBo , /tmp/Sbo/package-latex2html, /tmp

Es improbable que el directorio SBo no exista a menos que no se hayan ejecutado otros slackbuilds

en el pasado. /tmp debería estar ahí por defecto con la instalación de slackware.

cd $tmp : mueve el lugar donde Bash está trabajando de a /tmp/SBo

rm -rf $PRGNAM-$VERSION se deshará de cualquier entrada previa del directorio (tal vez fallida) para latex2html-2019.2.

tar xvf $CWD/$PRGNAM-$VERSION.tar.gz : Esto equivalía a desempacar de latex2html-2019.2.tar.gz , el cual estaría en el interior del slackbuild de slackbuilds.org llamado “latex2html”.

cd $PRGNAM-$VERSION : Este es el comando para “cambiar de directorio”. El shell bash funcionara ahora desde el contexto que se encuentra dentro de la fuente desempacada, que se encuentra en $CWD. Es decir, estamos desempacando el slackbuild pero dentro de la fuente desempacada.

chown -R root:root . : Note el punto, con un espacio al final de la línea; esto significa todo en el directorio actual. -R es recursivo. Así que aquí estamos dando propiedad a root y al grupo root.

```
 find -L . \
\( -perm 777 -o -perm 775 -o -perm 750 -o -perm 711 -o -perm 555 \
  -o -perm 511 \) -exec chmod 755 {} \; -o \
\( -perm 666 -o -perm 664 -o -perm 640 -o -perm 600 -o -perm 444 \
  -o -perm 440 -o -perm 400 \) -exec chmod 644 {} \;
```

Si entiendo correctamente el bloque de código anterior, está usando la “búsqueda” en base a los permisos y siguiendo los enlaces simbólicos usando “-L flag”. Si entiendo esto correctamente, es básicamente establecer los directorios a 755 para habilitar un “cd” en ellos y los archivos para que root pueda leer y escribir. Si esto es cierto, podría haber esperado:

```
-type d -exec chmod 775 {}
```

Para directorio y

```
-type f -exec chmod 644 {}
```

para archivos.

Próximo bloque de código:

```
CFLAGS="$SLKCFLAGS" \
CXXFLAGS="$SLKCFLAGS" \
./configure \
  --prefix=/usr \
  --libdir=/usr/lib${LIBDIRSUFFIX} \
 A test is applied and of course there will be a result depending on the
outcome. An example of a
bash "if" block statement is:
  --sysconfdir=/etc \
  --localstatedir=/var \
   --with-perl=/usr/bin/perl \
  --enable-eps \
  --enable-gif \
```

```
  --enable-png \
  --build=$ARCH-slackware-linux \
  --host=$ARCH-slackware-linux

make
make install DESTDIR=$PKG
```

Un par de cosas para decir aquí, el uso de "\" es una forma de usar un comando largo sobre varias líneas. Ya se mencionó CFLAGS y CXXFLAGS. A couple of things to say here , the use of "\" is a way of using a long command over several lines. We have already mentioned CFLAGS and CXXFLAGS. También hemos mencionado anteriormente la variable SLKFLAGS y aquí la usamos para establecer el valor de CFLAGS y CXXFLAGS.

En este script slackbuild latex2html también utilizamos un proceso de tres pasos de configurar, make, make install. Pero, ¿qué pasa con los tipos de –enable-eps, ¿de dónde viene eso?

Bueno, si tomas el código fuente de latex2html source desempaquetas (una forma rápida es hacer click con el botón derecho, abrir con Ark) ingresar al directorio y ejecutar:

```
./configure --help
```

Entonces obtendrá información útil de los desarrolladores. Te dice la opción y cómo puedes habilitar algunas de ellas.

```
make
make install DESTDIR=$PKG
```

Aquí, make, make install se llevan a cabo. Note que DESTDIR es una bandera que indica dónde irá el paquete.

$PKG equates to /tmp/SBo/package-latex2html

Proximo bloque de código:

```
find $PKG -print0 | xargs -0 file | grep -e "executable" -e "shared object"
| grep ELF \
  | cut -f 1 -d : | xargs strip --strip-unneeded 2> /dev/null || true

mkdir -p $PKG/usr/doc/$PRGNAM-$VERSION
cp -a \
 FAQ INSTALL LICENSE MANIFEST README.md TODO  \
  $PKG/usr/doc/$PRGNAM-$VERSION
cat $CWD/$PRGNAM.SlackBuild > $PKG/usr/doc/$PRGNAM-
$VERSION/$PRGNAM.SlackBuild
cp $CWD/manual.pdf  $PKG/usr/doc/$PRGNAM-$VERSION

mkdir -p $PKG/install
cat $CWD/slack-desc > $PKG/install/slack-desc

cd $PKG
```

```
/sbin/makepkg -l y -c n $OUTPUT/$PRGNAM-$VERSION-$ARCH-
$BUILD$TAG.${PKGTYPE:-tgz}
```

Las dos primeras líneas de este bloque están un poco llenas:

```
find $PKG -print0 | xargs -0 file | grep -e "executable" -e "shared object"
| grep ELF \
  | cut -f 1 -d : | xargs strip --strip-unneeded 2> /dev/null || true
```

Sin embargo, podemos elegir palabras clave que son órdenes y que pueden ayudar a darle algún sentido. “find” es una poderosa utilidad, con más de 50 opciones, localizada en /usr/bin/find. Básicamente hace lo que dice en la lata. Con la opción -print0 separa lo que encuentra con “\000” - en una palabra NULL. La tuberia “|” es usada para pasar los resultados de un comando a otro; en este caso xargs tiene un flag (bandera) -0. Esta opción es para que xargs acepte la entrada que tiene /000 entre ellos. ELF es un formato ejecutable y enlazable.

Para dar una respuesta sucinta las dos líneas están eliminando los símbolos de depuración y otras cosas innecesarias para que los binarios sean más pequeños, más rápidos y ocupen menos memoria.

```
mkdir -p $PKG/usr/doc/$PRGNAM-$VERSION
```

Aquí estamos preparando un directorio que se llamara “latex2html-2019.2” localizado en /usr/doc. Esto es así para que pongamos la documentación relevante en un directorio. De esta forma, un usuario puede acceder a la documentación del paquete. Las siguientes líneas ponen archivos como README.md en el directorio /usr/doc/latex2html-2019.2.

```
cat $CWD/$PRGNAM.SlackBuild > $PKG/usr/doc/$PRGNAM-
$VERSION/$PRGNAM.SlackBuild
```

Esa línea vuelve al directorio original que Latex2html. SlackBuild fue ejecutado desde (yo previamente ingrese al Desktop) abre el SlackBuild con el comando “cat” y lo copia al directorio de documentación.

Ahora, antes de enviar latex2html a slackbuilds, obviamente hice algunas pruebas y encontré que cuando el paquete fue instalado tenía una salida bastante completa de lo que podía hacer simplemente usando:

```
$ latex2html --help
```

También tuve acceso a un completo manual en formato pdf; así que en mi caso no escribí código para las páginas de manual. En su lugar, simplemente puse una copia de “manual.pdf” en el directorio /usr/doc/latex2html-2019.2.

— andy brookes 2020/01/05 16:29 (UTC)

Si enseñas matemáticas no te impide incrustar un poco de inglés.

Una plantilla en blanco puede ser obtenida de: https://slackbuilds.org/templates/

Fuentes

Estoy usando un script de SlackBuild que envié a slackbuilds.org: latex2html slackbuild.

- Escrito originalmente por andy brookes
- Traducido por — *rramp 2020/01/13 16:29 (UTC)*.

howtos