Anatomy Of a Slackbuild

Preamble

I guess all Slackware users will have used at sometime a SlackBuild script in order to create a package of software that could be easily and cleanly installed and removed later if needed. My experience was that I ran the SlackBuild script to create a package and never actually looked at the code it contained until one day the source version did not match that stated in the script.

Why would you bother looking at the code in a SlackBuild script? Well firstly you can use it if you are learning bash. Secondly you may not want to submit a SlackBuild script but perhaps you want to create a package that does not exists for software you want.

So here i would like to go through a SlackBuild script as best as I can (verbosely). I openly welcome edits from all others at all levels. In fact I think it important that people from all levels contribute. What is obvious to one person may need explaining to another person.

There are different approaches to a SlackBuild script and for instance Alien Bob has his very useful SlackBuild creator at : Aliens SlackBuild Toolkit

These days at SlackBuilds.org they prefer submissions using a SlackBuild template. I think it would be useful to go through a SlackBuild script which is currently on SlackBuilds.org. For this I will use the SlackBuild script I submitted for latex2html latex2html to SlackBuilds.org and which is currently available for Slackware 14.2 .

That way we are talking about a contemporary working SlackBuild script that you can actually download, use and hopefully relate to after reading this.

The other thing to mention is that the **context** of this page, when you read it, regarding a latex2html script is that a whole “slackbuild” from slackbuilds.org has been downloaded, is somewhere convenient (say on my desktop) that it is unpacked and into the unpacked directory called “latex2html” the software source called latex2html-2019.2.tar.gz has been placed.

Also that you have kick started the script :

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

Bash

In the days before Windows on Unix, a shell or Bourne Shell (Stephen Bourne, Bell Labs) was a way of communicating with the system. In a nod to Stephen Bourne , Brian Fox came up with a new version in 1989 and called it **B**ourne **a**gain **Sh**ell (bash).

If you open up a terminal in Slackware - say `Menu → System → Console` and type at the `$` prompt:

```
bash --version
```

, you should get something like:

```
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>
```

In simple terms , the input was received and acted upon. The input is somewhat short lived, since when you close the window the input has gone.

A bash script in simple terms is bash code that can be saved as a file on say a hard drive.

The first line of a bash script has this line:

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

This is commonly referred to as a *shebang*. This defines that the file is a shell script and also the path to the shell interpreter.

SlackBuild Script

We have so far described that a SlackBuild script is in essence bash code presented in a more permanent presentation of a file.

At the top is the *shebang*.

After that in line with slackbuilds.org requirements the next lines are the name of the package, who wrote the script and optionally a copyright notice is added.

I will attempt to go through the 14.2 latex2html slackbuild line by line. If there are blanks, it's a prompt for others to chip in. It also probably means I don't fully understand that bit.

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

If I look at the package I have installed on my current 64 Bit Slackware it's : `latex2html-2019.2-x86_64-1_SBo`. Now if you look at the first line above you will see PRGNAM (program name). This is a variable and its value is set to the string `latex2html`.

The version is related to the software source release; if you go to the latex source web page, you will see that there was a source release on June 5th 2019. Thus I simply named the version after that release. For those learning bash (I still am): on the second line `VERSION=` on the left assigns a value to the variable called VERSION.

When assigning a value to VERSION, why use `VERSION=${VERSION:-2019.2}` and not simply

VERSION=2019.2?
Take the expression ${VERSION}. This means “the value contained in variable VERSION”. The more complex notation ${VERSION:-2019.2} means “the value contained in variable VERSION, but if that variable does not yet have a value then use the default value of '2019.2'”.
So that second line simply boils down to VERSION=2019.2.
A value for VERSION can be set outside of the script: if you specified a value for VERSION on the SlackBuild command line or if it has been defined in your Bash environment.

In the same way the variable TAG is set to SBo and when the package is created this shows it's a “slackbuilt” package. If you look at packages in other repositories, you will see in the name eg chromium-77.0.3865.75-x86_64-1alien that the package is by Alien Bob.

The next block of code is as follows:

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

In computer programming languages such as php, python and others including bash there are common practices and logic. One common principle is the procedure of having a test in code. 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:

```
if[ test condition]
then
code to be executed depending on result
fi
```

In the above block, quite simply the “if” is the start of an If test condition and “fi” denotes the end of a If test condition.

Going back to bash for a minute you can set a bash variable like so

```
ARCH=something
```

and you can get the value of a variable and see what value the variable “ARCH” contains by putting a dollar sign in In front of it like so:

```
echo $ARCH
```

A -z flag can be used in an “if” statement to see if a string is empty. So lets have a look at the first line again and work out what it means.

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

In the above we don't need to see the value of ARCH; as long as we can access its value and be able

to use it in the script. So the first part before the “then” simply equates to , if the value of the variable ARCH is expressed as a string and its value is empty, do something. ARCH could represent a string value of the architecture of the PC the script is running on.

Another common algorithm is called in for instance php is the “switch statement” . This put in its simplest terms is a list and the test is to see if a certain value you have can be found in that list. If its not found you can place on the line last in the list would you would like to do, if nothing is matched.

In bash its in principle the same idea but its referred to as “case statement” .In principle you can put anything you like in the case block, but if you think about the fact we want to find out the “architecture” of a computer and already know there are only certain possibilities then it makes sense to try out a couple of known possibilities in the list first and if there is no match do something to get an answer. arm and i586 are of course two types of computer architecture.

Now if you try out this code in a terminal window:

```
uname -m
```

it should display the architecture of you PC, in my case its x86_64.

So to summarize regarding the block of code. First an “if” statement is run to see if the variable “ARCH” is empty. If there is a value for the ARCH variable nothing in the block of the “if” and inner “case” will run;but if an empty string is found (ARCH has no value) a “case statement” is run(within the if block of code) to find a match.If a match was found the variable ARCH would be set to the value of the match found, and the execution of the case would come to a stop. If the ARCH variable was empty and no match was found in the case list , then the bottom line comes int play which is:

```
uname -m
```

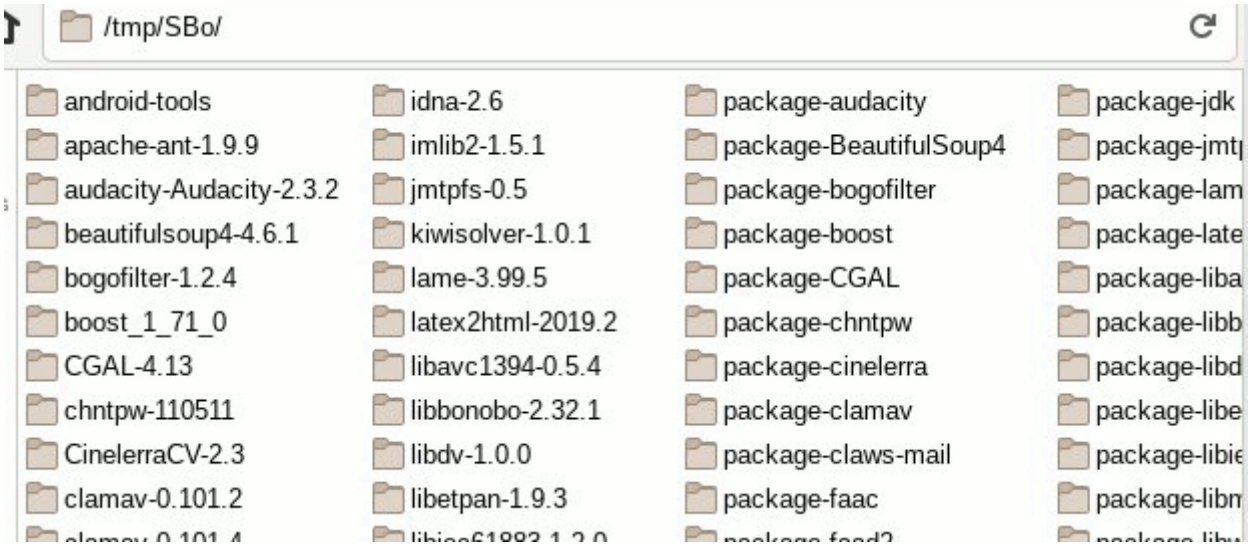
is used to get a result and ARCH set to the result.

Don't be fazed by the question mark in i?86 ,the question mark is a place holder that allows for possibilities via regex. it could be “3” (i386) , “6”(i686) etc.

Next block of code

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

Before we go into this let me have a look in my slackware file system and see whats there at /tmp/SBo.Taking a quick look at the image will give you a clue that the slackbuild works, by using the /tmp/SBo/ directory and creates a directory with the syntax package-packagename.So if we now have a look at the code above. CWD (current working directory) is a variable and is set to the value of pwd. If you run that in a terminal window, it will tell you where you in a bash context where you are working from.



TMP is going to be set to /tmp/SBo. Now lets have a look at

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

You might guess PKG is going to be set for latex2html to :

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

If you look closely at the image (taking into account /tmp/SBo/ at top of image) you will see exactly that in the image.OUTPUT is set to /tmp

Next block of code:

```
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
```

Probably we need, before we look at the rest of the code for latex2html slackbuild to introduce some basic concepts.

Historically computer software is installed in a three step process called configure, make, make install. Configure gets ready to build the software , see if everything needed is there and builds a new make file. A make file is a file that contains instructions to build a program.

You can from the command line install software just using configure, make and make install. A slackbuild does the job of creating a package so that the process of installing is more manageable

and reliable fashion. When you have a slackbuild downloaded on your system should there be a new release of source code its a simple matter of putting that source in your unpacked slackbuild and a quick edit to the slackbuild script.

Computer program source are written in “high level “ languages but are converted into a form that the computer can readily understand.Code written in the C language involves a C compiler converted it to binary.

The whole goal of any system installing a program, is that it should involve the concept of making it “tailor made” for the computer its being installed. Obviously that is going to involve the computer Architecture. During the compile process the system can be tweaked by passing in options from variables.

So lets now have a look at the block of code above. The block of code is simply an “if , else block” , where the code is executed top to bottom and amounts to -if the architecture is i586 set SLKFLAGS to .. if not go to next line.

CFLAGS and CXXFLAGS are variables holding values that can be passed in at compile time. We will see later that the variable SLKFLAGS will be used to set them.

Next block of code:

```
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: this stops the execution of the script if there is an error executing this code following this command
rm -rf $PKG: this deletes any previous directory (and contents) of package-latex2html at of /tmp/SBo/package-latex2html
mkdir -p $TMP $PKG $OUTPUT :mkdir with the “ -p ” flag creates directories, but only if they don't exist already.

that would be /tmp/SBo , /tmp/Sbo/package-latex2html, /tmp

Its unlikely that the SBo directory doesn't exist unless no other slackbuilds have been run in the past . /tmp should be there as default with the slackware installation.
cd $tmp : moves location where bash is working from to /tmp/SBo

rm -rf $PRGNAM-$VERSION will get rid of any previous directory entries(maybe failed) for latex2html-2019.2

tar xvf $CWD/$PRGNAM-$VERSION.tar.gz : This equated to unpacking of latex2html-2019.2.tar.gz , which would be inside the unpacked slackbuild from slackbuilds.org namely “latex2html”.

cd $PRGNAM-$VERSION : This is a “change directory” command. The bash shell will now be working from the context that is located inside the unpacked source, which is located $CWD. i.e We are back to the unpacked slackbuild but, inside the unpacked source.

chown -R root:root . : Notice the dot , with a space at the end of the line; this means all in current directory. -R is permission recursive. So here we are giving ownership to root and group 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 {} \;
```

If i understand the above block of code correctly its using the “find” on the basis of permissions and following symbolic links using the “-L flag”. If i understand this correctly its basically setting directories to 755 in order to enable a “cd” into them and files so that root can read write. If this is true I might have expected

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

For directories and

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

For files. Next block of code:

```
CFLAGS="$SLKCFLAGS" \
CXXFLAGS="$SLKCFLAGS" \
./configure \
  --prefix=/usr \
  --libdir=/usr/lib${LIBDIRSUFFIX} \
  --sysconfdir=/etc \
  --localstatedir=/var \
   --with-perl=/usr/bin/perl \
  --enable-eps \
  --enable-gif \
  --enable-png \
  --build=$ARCH-slackware-linux \
  --host=$ARCH-slackware-linux

make
make install DESTDIR=$PKG
```

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. We also previously mentioned the variable SLKFLAGS and here we use it to set the value of CFLAGS and CXXFLAGS.

In this latex2html slackbuild script we also utilize a three step process of configure, make, make

install. But what about the likes of –enable-eps , where does that come from ?

Well if you take the source code latex2html sourceunpack it (a quick way is right click , open with Ark) cd into it and run :

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

Then you will get some useful information from the developers. It tells you the option and how you can enable some of them.

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

Here, make, make install are carried out.Note DESTDIR is a flag to say where the package will go.

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

Next Block of code:

```
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}
```

The first two lines of this block are a bit of a mouth-full:

```
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
```

We can however pick out key words that are commands and that can help to make some sense of it. ind“ located at /usr/bin/find is a powerful utility that has around 50 options. It basically does what it says on the can.With the -print0 option it separates what it finds with “\000” - in a word NULL.

The “|” or pipe is used to pass the results of a command to another;in this case xargs which has a flag -0. This option is for xargs to accept input that has /000 between them. ELF is is Executable &

Linkable Format.

To give a succinct answer the two lines are removing debugging symbols and other unnecessary stuff to make the binaries smaller, faster and take up less memory.

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

Here we are preparing a directory which will be called “latex2html-2019.2” located at /usr/doc. This is so we an put relevant documentation into a directory relevantly called, so that a user can access documentation on the package. The next lines put files such as README.md into the /usr/doc/latex2html-2019.2 directory.

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

That line goes back to the original directory that Latex2html.SlackBuild was run from (i previously quoted Desktop) opens up the SlackBuild with “cat” command and copies it to the documentation directory. Now before I submitted latex2html to slackbuilds obviously I did some testing and found that when the package was installed it had a fairly comprehensive output of what it could do just using:

```
$ latex2html --help
```

Also I had access to a comprehensive manual in pdf format; so in my case I did not write code for man pages. Instead I simply put a copy of “manual.pdf” into the /usr/doc/latex2html-2019.2 directory.

— andy brookes 2020/01/05 16:29 (UTC)

If you teach maths it doesn't stop you embedding a little English.

Blank slackbuild templates can be obtained from : https://slackbuilds.org/templates/

Sources

I am using a SlackBuild script that i submitted to slackbuilds.org: latex2html slackbuild * Originally written by andy brookes

howtos