

HowTo Build a Package from Source the Smart Way

changelog

2009 September 11

- append "~bpo50+1" to the DEB's version number, to preserve the upgrade path.
 - (Thanks to [julian67](#) for the *debchange* suggestion).
- when using *dh-make*, rename the tarball to: `<package>_<version>.orig.tar.gz`
 - (Thanks to [infinitycircuit](#) for this suggestion).
- added some *checkinstall* tips.
 - (Thanks to [acimmarusti](#) for this suggestion).

2009 May 09

- original post

=====

This HowTo is a response to the unlimited supply of terrible advice that I have seen on GNU/Linux forums, on blogs and in various and sundry corners of the internet.

In particular, I wish people would stop advising others to use the *make install* routine. It's just as easy -- if not easier -- to build a DEB. Building a DEB gives you all of the advantages of package management, whereas the *make install* routine does not.

So this HowTo will describe several different ways that you can build a DEB package. Then -- for those of you who insist on using the *make install* routine -- it will explain how to use *make install* safely.

=====

Before we begin compiling however, let's review a few compilation basics.

First, you should have a reason for compiling package. There are many valid reasons why you might want to compile a package. The most common reason is to install a more recent version in Debian Stable. If that's your reason, then take a look around to see if someone has already built a DEB package for you (e.g. [Backports](#)). You'll save yourself a lot of trouble that way.

Another reason is to compile in support for a particular feature. For example, Debian's most recent version of Gnash uses FFmpeg as its sound handler, but I compiled the same version of Gnash with [GStreamer as the sound handler](#).

In the end, it doesn't matter what your reason is. Just make sure you have one and make sure it's a good one.

Second, if you're using Debian, then you'd be well-advised to use Debian-ized source instead of upstream source because it's very easy to build a DEB package if you're using Debianized-source.

If you build and install a DEB, then APT will be aware of the presence of the package and it will make sure that you do not overwrite the files of a previously installed package. By contrast, the

make install command will overwrite anything that gets in its way.

=====

Building a DEB is very easy to do and it's a shame that the members of this forum do not advocate it. If you've never built a DEB before, then try building [Backintime](#). It only takes a minute or two, so it's a great exercise.

In general, building a DEB from Debian-ized source is a simple four-step process.

1. First, you add a "source line" to your */etc/apt/sources.list* file and install the build dependencies.
2. Then you download the source (as **normal user**).
3. Then you run *dpkg-buildpackage* to build the DEB.
4. Finally, you install the DEB(s) with *dpkg -i*

To be more specific, add the following line to your */etc/apt/sources.list* file:

```
deb-src http://ftp.us.debian.org/debian/ sid main
```

Then install the build dependencies:

```
apt-get update
apt-get install fakeroot
apt-get build-dep <packagename>
```

Those are the last commands you should run as root before installing the DEB. All of the build commands should be run as **normal user**.

Next, create a directory for the build in your **normal user's** home directory and download the source into that directory:

```
mkdir /home/XXXX/my_build/
cd /home/XXXX/my_build/
apt-get source <packagename>
cd <packagename>-<version>/
```

If you're running Debian Stable, you may want to change the package's version number, so that the upgrade path to the next version of Debian Stable is preserved. To do that, you edit the *debian/changelog* file.

For example, I recently backported version 0.8.3a-1 of wxMaxima to Debian Lenny, so at this point, I ran:

```
debchange -b -v 0.8.3a-1~bpo50+1
```

A text editor then opened the *debian/changelog* file and I added an entry saying that I had

backported it to Lenny. After I saved the changes and closed the editor, the following warning message appeared:

```
debchange warning: new version (0.8.3a-1~bpo50+1) is less than
the current version number (0.8.3a-1).
```

That's exactly what I want -- a lower version number, so that the package is upgraded when I upgrade from Lenny to Squeeze.

Next, build the DEB:

```
dpkg-buildpackage -rfakeroot -us -uc
```

That last command may take a minute or an hour or three hours. It all depends on the size of the package and your own hardware.

Once the *dpkg-buildpackage* command finishes, you'll find the DEB(s) in your */home/XXXX/my_build/* directory and you can install them (as root) with:

```
cd /home/XXXX/my_build/
dpkg -i <packagename>_<version>_<architecture>.deb
```

It doesn't get much simpler than that.

Perhaps more importantly, a Debian Developer would follow a very similar procedure to create a backport from a package that's already in Testing or Unstable.

=====

If you're really eager to compile and there is no Debian-ized source in Unstable or Testing, you can still build a DEB package by using *dh-make*.

```
apt-get install fakeroot dh-make
```

In general, you begin by installing the build dependencies. If you're building a more recent version of package that's already in Debian, then the build-dependencies might be the same as the previous version. Otherwise, you'll have to figure out what they are (by reading the source package's documentation, for example).

Next, you extract the source into a directory in your home directory as **normal user**:

```
mkdir /home/XXXX/my_build/
cd /home/XXXX/my_build/
tar -zxf <packagename>_<version>.tar.gz
```

Because *dh-make* Debian-izes the source, you may need to change a couple of file names. For

example, suppose that we were trying to Debian-ize the upstream version of wxMaxima. In that case, we would have to change the name of the directory (to lowercase) and the name of the tarball (to the format: `<package>_<version>.orig.tar.gz`):

```
mv wxMaxima-0.8.3a/ wxmaxima-0.8.3a/
mv wxMaxima-0.8.3a.tar.gz wxmaxima_0.8.3a.orig.tar.gz
```

Next, enter the `<packagename>_<version>/` directory and run `dh_make`.

```
cd <packagename>_<version>/
dh_make
```

`dh_make` will then ask you if you want to create a: "single binary, multiple binary, library, kernel module or cdfs" and ask you to complete the package maintainer fields.

You might then have to tweak the `debian/rules` file (e.g. to set configuration options) or set a shell variable. It all depends on the package in question.

Once you have done that (or accepted all of the defaults), then the next step is to run (as **normal user**):

```
dpkg-buildpackage
```

and then install the DEB package (as root) with:

```
dpkg -i <packagename>_<version>_<architecture>.deb
```

For a good example of how to use `dh-make`, see: [HowTo Compile PDFedit](#).

=====

Debian-ized source should be your first choice when building from source. Use `dh-make` when Debian-ized source is unavailable because it Debian-izes the source for you.

If for some reason Debian-ized source is not available and `dh-make` doesn't work, then your third choice should be to use [checkinstall](#).

```
apt-get install checkinstall
```

`Checkinstall` is not as good as the first two options, but at least it produces a DEB, so it will prevent you from unknowingly overwriting system files and it will make APT aware of the package's presence.

In general, using `checkinstall` is very similar to the `make install` routine. The only difference is that you don't run the `make install` command.

First, you pick an installation directory. It's a good idea to pick a directory where no other packages

are installed or will be installed. For example, */opt/* or */usr/local/* are two good choices, but because *checkinstall* will build a DEB, APT will make sure that you don't overwrite any files even if you choose */usr/* (or if you let the defaults choose */usr/* for you).

Suppose you pick */usr/local/*. The next step is to pass that choice to the *configure* script. For example:

```
./configure --prefix=/usr/local --<configuration-options>
```

Then you would run:

```
make
su -c checkinstall -D make install
```

The trouble with *checkinstall* is that it does not Debian-ize the source or understand package dependencies. It simply keeps track of the files that the DEB contains.

It also annoys me that the files are installed at the same time that the package is built. (That's why you have to become root to use the *checkinstall* command). I wish it would just build the DEB and then let me decide when I want to install it.

Nonetheless, it does produce a DEB, so the package can be easily removed and won't overwrite other files without first obtaining your permission. Just remember to *chown* it back to your normal user:

```
chown XXXX:XXXX <packagename>_<version>_<architecture>.deb
```

EDIT: [acimmarusti](#) suggests using the command:

```
su -c 'checkinstall -D --install=no'
```

In theory, that command should simply build the DEB without installing it. In practice, it may or may not install the package.

While updating this HowTo, I tried his suggestion (with wxMaxima 0.8.3a) by running:

```
./configure --prefix=/opt
make
su -c 'checkinstall -D --install=no'
```

and unhappily discovered that some of the files were installed without informing APT of the changes.

Specifically, *checkinstall* installed the */opt/* directory's files, but did not install the */usr/share/doc/wxmaxima/* directory's files.

=====

If an RPM is available, you might want to look into [alien](#).

```
apt-get install fakeroot alien
```

It's just as bad as *checkinstall* (because it doesn't understand package dependencies), but it's not much worse and -- most importantly -- it produces a DEB.

To use it, you simply download the RPM and run as **normal user**:

```
fakeroot alien -d <packagename>-<version>.<architecture>.rpm
```

and then become root to install the DEB that *alien* produces.

```
dpkg -i <packagename>_<version>_<architecture>.deb
```

=====

Finally, there's the *make install* routine. *make install* should be your absolute last resort. You should **NEVER** use it unless all other options have failed and you should **ONLY use it IF** the package you want to build is of mission critical importance.

The *make install* routine is the most primitive method of building a package from source and has absolutely no concept of dependencies or package management. There's a reason why GNU/Linux distributions use package managers like APT or RPM. And that reason is to get as far away from *make install* as possible.

So if you're going to use *make install*, then at least use it intelligently. Do **NOT** blindly follow the lemmings running off the cliffs of the internet.

As with *checkinstall*, the most important thing to do is to pick an installation directory where no other packages are installed or will be installed. Once again, */opt/* and */usr/local/* are two good choices.

As with *checkinstall*, the next step is to pass that choice to the *configure* script. Suppose (again) that you pick */usr/local/*. You would pass that choice to the configure script with the *--prefix* flag:

```
./configure --prefix=/usr/local --<configuration-options>
```

That will cause the files to be installed in your */usr/local/* directory, which Debian doesn't install to by default. This should ensure that you don't overwrite important system files when you run *make install*.

The next steps are to build the package and install it:

```
make
su -c make install
```

But that's not the end of the story. If you want to uninstall a package that you installed with the *make install* routine, then you had better make sure you don't accidentally delete that package that you compiled (even if it grows to 20 GB) and you had better hope that the its *make uninstall* routine works just as well as its installation routine or you'll be stuck manually deleting all of the files.

That's why you should **ALWAYS** build a DEB package. Let APT remember where the installed files are. You've got better things to do ... like play with your newly installed package!

=====

In summary:

- It's easier to create DEBs from Debianized source than it is to use the *make install* routine.
- When Debianized source is unavailable, use a helper application to build a DEB, so that you can take advantage of APT's package management features.
- If you are unable to build a DEB package, then stop and ask yourself if you really need the package you're trying to build.
- You should **ONLY** use the *make install* routine as a LAST RESORT for MISSION CRITICAL applications.
- If you do use the *make install* routine, do it intelligently.

Good Luck and Have Fun!,
Soul Singin'