Rolling your own Debian packages

GIFT WRAPPED

Just a couple of steps are all it takes to convert a source code archive into a full-fledged Debian package.

BY CARSTEN SCHNOBER

The compiler was once regarded as an exotic tool for hard-core developers, but building programs is now a common task for average Linux users. The three classic commands *configure*, *make*, and *sudo make install* will help you convert the source code downloaded from a promising project website into a binary application ready to execute on your own system.

The problem is, if you build a program yourself, you won't be able to manage, track, and update the application through Ubuntu's handy package management system.

If you can't find an Ubuntu package for the program you want to install, but you still want the chance to manage the program using Ubuntu's package management tools, you can always convert the program into the Debian package format used with Ubuntu. In this article, I'll walk you through the steps required to convert a program into the Debian package format.

CheckInstall

The CheckInstall [1] program (Figure 1) offers the easiest approach to creating the DEB packages used with Ubuntu, as well as RPM packages used with other Linux variants. CheckInstall is easy to use, although it tends to fail with packages that extend the operating system functionality by adding kernel modules. If you prefer DEB packages that will run on systems other than Debian, you need more than the basic packages created by CheckInstall. Debian-based distros offer a collection of tools, mainly found in the *dpkg-dev* package, that facilitate the task of packaging an application.

Original Beats Imitations

Debian packages are quite easy to build as long as the source code directory is ready. All the required information must be located in the *debian* subdirectory of the source code directory. If the distribution has a prebuilt DEB package with the software you want to compile, the information is probably already available. To recompile this version – for example, with support for additional features – first download the matching source code using *apt-get source Packagename* (without *sudo*). You might need to enable the *Source code* package source in Synaptic.

After you download the files to a directory, you will need to change to this directory to run the command. You should have three new files and a directory. The filenames comprise the software name, the package version number, and possibly an internal revision number, such as *ubuntu2*. One of the three files is the original archive with the *.orig.tar.gz* suffix. The file with the *.diff.gz* suffix contains the changes the

distributor has made to the original source code to adapt the package to match the system. Then there is a *.dsc* file containing a description with details of the supported processor architectures and the package maintainer. The directory created by *apt-get source* also contains the *debian* directory.

In the simplest case, you could now just make any changes you need; for example, adding a program library to the software to extend its functionality. You can add more features to the source code or install patches with changes. To create a new Debian package, issue the *dpkg-buildpackage* command in the source code directory (the command is part of the *dpkg-dev* package).

To build a package using this approach, you actually need administrative privileges; however, if you install the *fakeroot* package, *dpkg-buildpackage* will automatically use it if you try to run the program without administrative privileges. Fakeroot lets you build Debian packages with write privileges for the source code directory and its parent directory.

First, the program checks to see whether the tools and libraries it needs for building the package exist on the system. If not, it issues a list of missing packages you need to install. Otherwise, the script will call *configure* and then *make* before finally creating a DEB package that it stores in the parent directory. To install the package, you need to launch the package manager; at the command line, the following command handles this:

```
$ sudo dpkg -i ↵
package_name.deb
```

Documenting the details is a good idea. The *changelog* file in the *debian* directory is the place to document your work,

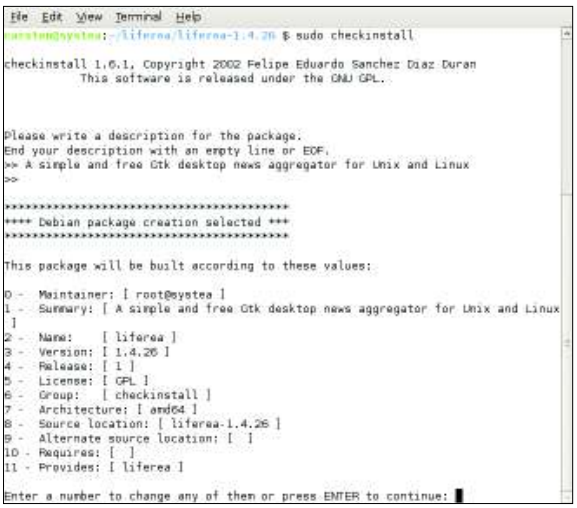

Figure 1: CheckInstall collects information about the program, but it does not define any dependencies.

but make sure you stick to the required format; otherwise, your package-building activities are doomed to failure.

Updates

In real life, program packages included by distributors often do not come with

Table 1: Files in the debian Directory

Filename	Filename and Path after Installation	Function
README.Debian	*/usr/share/doc/package_name/README.Debian*	Documents differences between the original source code and the "debianized" version
changelog	*/usr/share/doc/package_name/changelog.Debian.gz*	Logs the changes made to the package
copyright	*/usr/share/doc/package_name/changelog.Debian.gz*	License under which the software in the package is released
control	–	Information for the package: name, version, dependencies, description, etc.
rules	–	Rules for creating the package
conffiles.ex	–	Configuration files contained in the package
cron.d.ex	*/etc/cron.d/package_name*	Actions and programs that the program tells cron to execute regularly (in cron format)
dirs	–	List of directories required by the program, without leading slashes; as in */usr/bin*
docs	–	List of files to be copied to the */usr/share/doc/package_name/* directory
emacsen-install.ex	–	Emacs-specific (optional)
emacsen-remove.ex	–	Emacs-specific (optional)
emacsen-startup.ex	–	Emacs-specific (optional)
init.d.ex	*/etc/init.d/package_name*	The init script for starting and ending any system services contained in the package
manpage.1.ex	*/usr/share/man/man1/program_name.1.gz*	Man page for the program
menu.ex	*/usr/share/menu/package_name*	Information for a program entry in the start menu
watch.ex	–	Configuration file to update the package using *uscan* and *uupdate*
preinst.ex	–	Contains the commands the package manager executes before installing the package
postinst.ex	–	Contains the commands the package manager executes after installing the package
prerm.ex	–	Contains the commands the package manager executes before removing the package
postrm.ex	–	Contains the commands the package manager executes after removing the package

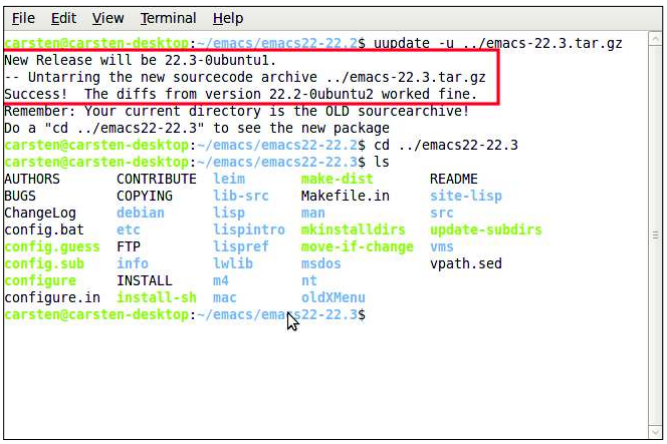

Figure 2: The uupdate command unpacks a source code archive and creates a debian subdirectory based on the content of the existing package.

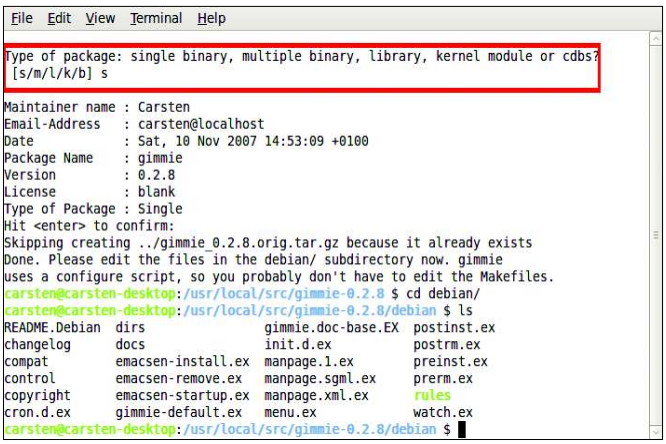

Figure 3: The dh_make command Debianizes a source code directory, creating a debian subdirectory where it creates and generically populates files.

the latest version; you need a new DEB package based on the source code for the updated program version. The *uupdate* command from the *Devscripts* package helps you update the source code.

To use *uupdate*, download the tarball with the new archive, but do not unpack it. Then run *apt-get source program* (again without *sudo*) to download your distributor's version of the same package. This will give you both a complete *debian* directory with information you can recycle, as well as (typically) the changes made by the distributor to modify the system for your distribution.

Now change to the *program version* directory created by *apt-get source* and run the *uupdate -u* command, adding the filename of the archive containing the new version of the program (Figure 2):

```
$ uupdate -u
path/program-NewVersion.tar.gz
```

The command reads the changes made by the distributor from the *program.diff.gz* file and attempts to add them to the new version. This approach will only work if the modified files are not much different from their counterparts in the previous version. If just a couple of the patches fail, the update process will carry on without them. *uupdate* also transfers information, such as the package description and dependencies, from the original package to the new version.

The call to *uupdate* unpacks the tarball with the updated program version, which contains a new source code directory, including a *debian* subdirectory with the package information gleaned from the original package. Next, run *dpkg-buildpackage* in the new source code directory, as described previously. This step starts the build, which gives you a full-fledged DEB package. Again, the package management tool is required to install the package:

```
dpkg -i package_name.deb
```

New Buildings

If you do not have a previous version of the program to build on, your only option is a completely new DEB package. To start, unpack the source code archive, which will not contain any distribution-specific information. To "Debianize" the source code, you must first install the *dh-make* package. Then run *dh_make* with the *-f* parameter in the source code directory, passing in the path to the original source code archive. So that other users can contact you, you also need to enter an email address for the package, which is defined with the *-e* parameter:

```
$ dh_make -e mail@address.com -f
../program-version.tar.gz
```

Next, you must decide what type of package you are building (Figure 3). In the easiest case, you enter *s* for a *single binary* package. The other options are *m* for *multiple binary*, *l* for *library*, and *k* for *kernel module*.

The CDBS (*b*, Common Debian Build System, [2][3]) option is useful mainly for more complex packages that you will want to use and update over an extended period of time; however, this option does require a deeper understanding of the DEB package system.

debian Directory

After you run *dh_make*, take a look at the files in the *debian* directory (Table 1). The files created here contain the placeholders and generic information you will need to modify to match the package.

The *control* file plays a fundamental role because it contains all the critical package information. You can enter the software category in the *Section:* line; Table 2 shows your options.

In many cases, an additional *Priority:* line is useful. The *Priority:* line describes how important the software is for the

Table 2: Software Categories for "Section:"

Designation	Purpose
admin	System management programs
base	Basic packages
devel	Packages for software developers
doc	Documentation
libs	Program libraries
mail	Email software
net	Network software
x11	Other graphical programs

INFO

[1] CheckInstall: *http://asic-linux.com.mx/~izto/checkinstall*

[2] CDBS: *http://build-common.alioth.debian.org*

[3] Debian packages via CDBS: *http://www.ngolde.de/cdbs.html*

system; in the case of packages you build yourself, the priority will typically be *optional* or *extra*. The dependencies are specified in the *Depends:* line, where *dh_make* has already created an entry:

```
${shlibs:Depends}, ➥
${misc:Depends}
```

These variables save you some work; when you compile, the *dh_shlibdeps* program automatically replaces the variables with the required packages.

If you want to add recommended packages, insert a *Suggests:* line into the *control* file;

```
Suggests: libqt0-ruby1.8
```

The packaged program will not strictly require the programs and libraries stated in the *Suggests:* line, but in many cases, the recommended packages will enable features that users would otherwise have to do without.

All that remains is to fill out the program *Description:* field. The first line should contain a short description of no more than 60 characters; below it, there is enough space for more details. Indent each line with a blank so that the package builder will know that the lines belong to the *Description:* field.

Home Straight

The *rules* file is another important inhabitant of the *debian* directory. The *dpkg-buildpackage* command, which finally triggers the package build, takes its instructions from the *rules* file. For simple programs, the rules generated by *dh_make* will work without modification. If you are packaging more complex software or program libraries, you can add parameters, such as compiler options, to the *rules* file.

The *copyright* file contains the names and email addresses of the programmers and copyright owners, and the license under which the software package is published. Indent the text with four blanks. If you are using GPL licensing, you don't need to add the whole text, since it is available on any Debian-based system in the */usr/share/common-licenses/GPL* directory. In this case, you can simply add a reference to the license text in *copyright*.

Checking the *changelog* file for any new package is a good idea. The first time you build a package, you normally won't need to change the log and can just delete the file. In all other cases, you will need to edit the file. The *README.Debian* file is where you add instructions that relate to the DEB package in general and not to the software it contains.

Small Steps

If you build software yourself, the small effort of building matching packages will give you enormous convenience benefits. The *uupdate* utility updates on the basis of an existing package, and *dh_make* creates the *debian* directory. With just a little practice, a final call to *dpkg-buildpackage* quickly becomes routine.

Seeing your first home-grown package install is a really great feeling! ■