
Contents

1. [Introduction](#)
 2. [Requirements](#)
 1. [configure apt](#)
 2. [create a working directory](#)
 3. [Choose the package](#)
 3. [The packaging workflow](#)
 1. [Get the source package](#)
 2. [Get the build dependencies](#)
 3. [Rebuild without changes](#)
 4. [Edit the source code](#)
 5. [Building the modified package](#)
 6. [Installing and testing the modified package](#)
 7. [Building the source package](#)
 8. [Sending your changes to the BTS](#)
 4. [Conclusions](#)
 1. [That's it, get ready for the next package](#)
 2. [Feedback](#)
 3. [Questions and Answers](#)
 4. [See also](#)
-

Introduction

This tutorial is based on a  [Debian Women Build it Event](#) held by Margarita Manterola (in collaboration with the  [OpenHatch project](#)) on 07-May-2011

This tutorial is about taking an existing package, re-building it, applying changes to it, and preparing those changes so that you can send them to a bug as a patch.

Requirements

You need very little previous knowledge for this tutorial,

just no fear of the command line 😊

Technical requirements:

- You should have a working Debian distribution or a Debian based distribution.
- You should have administration rights in this computer (either root or [DebianPkg: sudo](#)). Every time that admin rights are needed, we'll include sudo in front. If you don't use sudo, just get the rights whatever way you like.
- [DebianPkg: build-essential](#)
- [DebianPkg: fakeroot](#)
- [DebianPkg: devscripts](#)

Remember:

```
sudo apt-get install build-essential fakeroot dev
```

configure apt

Once you have installed the needed packages, the next thing that you need to do, is make sure that you have some **source repositories** configured in your computer.

Open your **/etc/apt/sources.list** file and check if you have one or more lines that start with **deb-src**.

```
deb-src http://httpredir.debian.org/debian unstable m:
```

This line is needed in order to work with source packages.

If you don't have any *deb-src* lines, you'll need to add at least one . This is usually achieved by copying one of the

existing *deb* lines and changing it to *deb-src*. You can do that by running an editor with admin rights (`sudo gedit`, `sudo kate` or `sudo vim`).

Usually, it's a good idea to use **unstable** as the repository, since you'll be working with the latest version of the package. But if you intend to modify a package as it is in **stable** or **testing** you could use those distributions as well.

If you use stable/testing/whatever as your running distribution, getting source from unstable won't affect it.

Once you've added the line, you'll need to do

```
sudo apt-get update
```

in order to update the list of packages available for installation.

create a working directory

With the sources URL added to your apt repositories, you'll now be able to get the source of **any Debian package that you like**.

For this particular tutorial, we are going to download the source of one package and make a small modification to it, so that it works better.

It's always a good idea to have a directory that you use to work with source software, separated from other directories used for other stuff. In case you don't already have one, I'd suggest that you create a directory *src* with another one called *debian* inside it:

```
mkdir -p src/debian/; cd src/debian
```

Inside this directory we will get the source of the package that we want to work with.

Choose the package

In this example, we'll use a package called [DebianPkg: fdupes](#), a tool to detect duplicate files, and we will be fixing the ~~Closed: #585426: fdupes: Strange help message: Debian bug 585426.~~

You should install the package (or check if you have it installed and up to the latest version) before proceeding, since you'll need to have the dependencies sorted up when you want to install the modified one.

If you don't have [DebianPkg: fdupes](#) installed, you can do this by doing:

```
sudo apt-get install fdupes
```

and check that the bug is still present. You can do that by running

```
fdupes --help
```

and checking that the second line of info for the *--debug* option still doesn't make any sense.

The packaging workflow

Get the source package

In order to **get the source** of [DebianPkg: fdupes](#), what you need to do is go to your chosen directory (src/debian in this example) and do (as normal user):

```
apt-get source fdupes
```

You have now downloaded the 3 files (*.dsc*, *.tar.gz*, *.diff.gz*), composing the **Debian source package**.

It probably also informed you that [DebianPkg: fdupes](#) is maintained with *Git*.

Once the package is downloaded, you can check the directory where you are (typing `ls`), and you'll find that apart from the 3 files that were downloaded you also have a directory, called **fdupes-1.50-PR2**. This is the **unpacked source** of the Debian package.

To enter that directory, type:

```
cd fdupes-1.50-PR2/
```

When you check the contents of this directory (typing `ls` again), you'll see quite a number of files of different sorts, and a **debian directory**.

Every Debian (or Debian derivative) package includes a *debian* directory, where all the information related to the Debian package is stored. Anything that's outside of that directory, is the **upstream code**, i.e. the original code released by whoever programmed the software.

Go into the **debian** directory, by typing

```
cd debian
```

This is the directory that the package maintainer has added to the source code to build the package.

In this directory you'll usually find lots of files related to Debian's version of the program, Debian specific patches, manpages, documentation, and so on. We won't be going any deeper about these files here. Look at its contents by typing `ls`.

Just keep in mind that

- the **rules** file is the executable file that we will be running in order to build the package.
- in the **patches** directory, there are also a number of patches applied by the maintainer

Let's move one directory back, by doing

```
cd ..
```

You should be again at **fdupes-1.50-PR2**, the main directory of the source code.

Get the build dependencies

In order to build almost any program, you will need some **dependencies** installed.

The dependencies are the programs or libraries needed to compile your program. Usually it's a bunch of packages that end in *-dev*, but it might also be other things like [DebianPkg: automake](#) or [DebianPkg: gcc](#), depending on how many development tools you've ever installed in that machine.

[DebianPkg: apt](#) provides a way of easily installing all the needed dependencies:

```
sudo apt-get build-dep fdupes
```

Once you've downloaded these tools, you'll be ready to build the package.

Rebuild without changes

Before making any changes to the code, let's **build the package** as it is right now, just to make sure that it builds and it installs properly. Do:

```
debuild -b -uc -us
```

The extra parameters are to build a binary only package (.deb) and prevent it from signing the package, since we don't need to sign it right now.

This command will probably take a while to run, since usually it first has to run `./configure`, then it has to compile the source code and then build the packages. In fact, it will run the commands that are listed in the *debian/rules* file and will hopefully end with a message like:

```
dpkg-deb: building package `fdupes' in `../fdu
```

In your own language. `<your arch>` can be *i386*, *amd64*, depending on which architecture you are running your machine on.

Once the package has correctly built, the next step is to **install this file** with:

```
sudo dpkg -i ../fdupes_1.50-PR2-3_<your arch>.deb
```

After that, **check that the bug is still present**, running

```
fdupes --help
```

Edit the source code

Now, we want to actually **fix this bug**.

Here comes the fun part. When you are trying to fix a package bug, sometimes it will be located in the upstream source, sometimes it will be related to how the program was packaged for Debian. So you'll be editing different files depending on where the problem is.

In this particular case, the package uses the [DebianPkg: dpatch](#) tool, a tool to manage patches for the package, so we will make use of that tool. Other packages use a different tool, called [DebianPkg: quilt](#), to manage patches, but we won't be covering that one in this tutorial.

To create a new patch, you'll need to do the following.
Type:

```
dpatch-edit-patch 80_bts585426_fix_help 70_bts537
```

This will start a new shell inside a special environment, where you can edit your files and *dpatch* will afterwards take care of getting the differences with the original.

- the first parameter is the name assigned to the new patch : 80_bts585426_fix_help
- the second parameter is the last patch that should be applied before applying the new one :
70_bts537138_disambiguate_recurse.dpatch

The name of the patch was chosen to match the pattern already established by the maintainer.

Now we need to edit the **fdupes.c** file. Go to the line 1066 and delete it. The line says:

```
printf("                \teach set of duplicates wi-
```

You can edit the file with your preferred editor (vi fdupes.c, gedit fdupes.c, kate fdupes.c, etc).

Once you are done, you should type into the console :

```
exit
```

It will end the special environment that *dpatch* created for us, and you'll have a new patch in the **debian/patches/** directory. Check it out with:

```
cat debian/patches/80_bts585426_fix_help.dpatch
```

In order for this patch to get applied, you'll need to edit the **debian/patches/00list** file, and add after the last line :

```
80_bts585426_fix_help
```

The *00list* file is the *dpatch* file that lists all the patches that will be applied. They are applied in order, from the one appearing in the first line, till the one appearing in the last line.

Before rebuilding the package with this patch, we want to **make our package different from the original one**, so that we can afterwards extract the changes in order to send them as a patch to the bug. In order to do this, type:

```
dch -n
```

This will

- add a new entry in the *changelog* file, maybe with your name (depending on other configurations that we are not going to cover), with the current date,
- and open the changelog with the configured command-line editor.
 - In case this is vi, and it's your first time with vi, you can start editing by pressing the *Insert* key, and after you are finished, you can save and close by pressing: *ESC :wq*

So, you are editing the **changelog** file now. What you have to enter in this file is some description of the change that we've made. For example:

```
Added a patch to fix the --help issue with the --debug
```

Do this in the line with the empty *.

Building the modified package

Once this is done, just build the package again with the same command as before:

```
debuild -b -uc -us
```

Add `-tc` option if you want to clean build artifacts.

In the case that you need to debug the compiled package, particularly if it's a *Segmentation Fault* that you are trying to fix, you might want to compile it like this, so that the code is not optimized and not stripped and it's easier to debug:

```
DEB_BUILD_OPTIONS='nostrip noopt debug' dpkg-build
```

You'll see a some compiling output on screen. This is usually not very interesting, unless you are looking for a bug that is related to the compilation of the package itself. Normally, I just let this go while I do something else (grab some cookies for your coffee, for example).

This time, the package created should be:

`../fdupes_1.50-PR2-3.1_<your arch>.deb`, the version changed because dch changed it for us in the changelog (-3.1 instead of -3).

Installing and testing the modified package

Install it with:

```
sudo dpkg -i ../fdupes_1.50-PR2-3.1_<your arch>.d
```

and **test** that the help is now correct. 😊

If in any case what you've done has made things worst, you can always revert to Debian's version by doing:

```
apt-get install --reinstall fdupes=<previous_version
```

Building the source package

Once a bug is fixed, you might want to also **build the source package**. This means not only the .deb file, but the other files that we downloaded at the beginning. This is done with:

```
debuild -us -uc
```

If you've built the source package, go to the previous directory with `cd ..` and check the files there with `ls`. You'll see that you have more files there now, including 2 *dsc* files, one for the original package and one for the one you just made.

Sending your changes to the BTS

Once you've built any source package, you can **find out the difference between your package and the original one**, by using [DebianPkg: debdiff](#):

```
debdiff fdupes_1.50-PR2-3.dsc fdupes_1.50-PR2-3.1
```

In this particular case, since we used the *dpatch* tool, what we would send to the BTS as a patch is the *dpatch file* that we created, because the change that we made is enclosed there.

But if we hadn't used *dpatch*, we could use the output of that *debdiff* and send that to the BTS.


Conclusions

That's it, get ready for the next package

You're done with modifying the package, you can now keep fixing bugs in other Debian packages! These are the important commands you'll need to remember:

```
apt-get source the-package
apt-get build-dep the-package
(...)
fakeroot debian/rules binary
dpkg -i the-package-blabla.deb
dpkg-buildpackage -us -uc
debdiff package-blabla.dsc package-blabla.1.dsc >
```

Feedback

I hope you enjoyed the tutorial, please  [send me your feedback](#) if you feel like it, or modify this wiki if you want to add some useful information I might have left out.

Questions and Answers

QUESTION: Is the number 80_ in the name of the patch a cosmetic thing?

ANSWER: It's so that you can get them in order when you list them with *ls*. The order of application is the order in which they appear in the *00list* file. Some maintainers choose to give them numbers, some don't. But when you are patching a package, you should try to follow whichever pattern the maintainer chose.

QUESTION: About dpatch-edit-patch params, where did you get your params? I mean the bts****

ANSWER: The first one is the name of the patch. I chose a name similar to the other ones. "bts" stands for "bug tracking system", the number is the bug number that we are closing.

QUESTION: How does dch pick the number "3.1"?

ANSWER: We used dch -n, because we are not the maintainer of the package (n stands for non-maintainer). Then it's 3.1. If you were the maintainer, you'd have done dch -i, and it would have chosen -4

QUESTION: When sending a patch to the BTS, do we send anything about the changelog?

ANSWER: Not necessarily, but you can send that too, it depends on the bug that you are fixing. Sending the debdiff output is always acceptable.

If you're sending only a small and simple patch it's normal to let the maintainer write the changelog; if it's a big NMU update then yes you should send the changelog as part of your big patch to the BTS.

QUESTION: Sending a changelog with a NMU is considered not polite?

ANSWER: What some (not all) maintainers don't like is if when someone actually uploads an NMU. Sending a patch to the BTS is always/immer/siempre/sempré well received!

QUESTION: What is the quickest/easiest way to find out which tool (if any) is used to manage patches?

ANSWER: If there's a build dependency on quilt or dpatch in debian/control. Or if there's a debian/README.source

file.

QUESTION: Does this workflow work for all packages?

ANSWER: The workflow is quite general. The only thing that might change is the patch management system (dpatch/quilt/none). The rest is basically always the same.

See also

- If you want to go on a little bit further in your package making, you can read some of the [AdvancedBuildingTips](#)
- You can find a tutorial about creating new packages for Debian at [IntroDebianPackaging](#)
- [Packaging](#) is the page that gather everything about packaging on this wiki