

Packaging with GIT

Pierre Habouzit

FOSDEM 2008 - Debian Room

23 Feb. 2008

- 1 Why choosing GIT
- 2 Repository layout
- 3 How to turn mud into gold efficiently
- 4 The END!

Because SVN isn't good enough

I've packaged with SVN on svn.debian.org for a long time... **well, it sucks:**

- I wasn't able to work without network access;
- I couldn't work incrementally: you have to get other's patches;
- I couldn't "try and see" and maybe discard some work without reverting patches: SVN never forgets;
- I like having the full upstream source at hand, but svn explodes doing that for large sources (KDE ...).

Because GIT does what I need

With GIT, as a DSCM, most of the issues are gone.

- Off-line work is possible, in fact you always work off-line;
- Incremental work is possible;
- GIT allows you to completely erase a wrong idea if you didn't share it yet;
- GIT storage is extremely efficient: the marginal cost of commits decreases with the number of commits.

A bit more about GIT storage

GIT storage is very efficient and optimized. Some numbers:

xorg-xserver.git, goes back to 2000, is **20MB** big. The last orig.tar.gz is 8MB big, more than 84MB unpacked.

dpkg.git, whole history since April 1996, generates a git pack of **15MB**. The last dpkg release is 17MB big unpacked.

GNU libc version 2.7 weights 115MB unpacked. The full glibc history (starts in the eighties) generates a GIT pack of **104MB**.

Though, this won't probably be true for packages with a lot of binary stuff in it, where delta compression is less likely to produce good results (games data packages come to mind).

A real maintainer toolbox

GIT is special, because it was designed by a **Maintainer**. This has several implications:

- GIT operations to look at the history are blazingly fast;
- GIT operations to integrate patches are blazingly fast;
- GIT supports advanced merging features;
- GIT has advanced patch manipulation features.

In other words, GIT is a suitable replacement for both SVN and quilt¹.

¹or your preferred patch system

The repository layout: upstream

First of all, my repositories contain upstream sources.

- 1 if upstream uses a public SVN or GIT repository, I track their SCM under the `upstream/*` namespace.
- 2 if upstream tarballs differ from the SCM, I also add a `upstream/tarballs` branch that replicates the tarballs directly.
- 3 if upstream only releases tarballs, I only have an `upstream` branch that is the successive import of tarballs.
- 4 last and not least, `pristine-tar` helps me saving the pristine upstream tarballs inside git.

The repository layout: upstream

Demo: tokyocabinet packaging:

- Step 1: importing upstream;
- Step 2: backing up the orig.tar.gz.

The repository layout: patches

Second of all, instead of using quilt, I use a private git branch to hold my patch queue (usually called `${upstream_branch}+patches`). There are many reasons to that:

- I only use one tool: GIT;
- quilt doesn't know about diff3, unpractical for partially merged patches;
- I believe it to be nicer to browse or to cherry-pick for upstreams.

Demo: let's rebase our patch branch for tokyocabinet!

The repository layout: debian packaging

Finally, everything that is Debian specific is kept into a `debian` branch.

- One branch per suite: `debian-etch`, `debian-sid`, `debian-exp`;
- the corresponding upstreams are merged into this `debian` branch;
- the `upstream+patches` branch is serialized under `debian/patches`.

Demo: let's release our `tokyocabinet` package!

Hiding to the world you're a dirty pig

There is nothing more useless than a crappy SCM history. I don't know how *you* work, but I tend to do everything at the same time.

Well, then just bind `:wa` to `:wa<cr>:!git commit -a` in your editor...

Then work like a pig, compulsively saving[^]Wcommitting your stuff...

And when you're happy of the current state, let's pretend that you're a good boy.

Demo: the same in pictures.

Using GIT for other's packages

GIT is not only useful as a versioning tool, it also helps to work fast with NMUs and security uploads.

- 1 `$ git init && git add . && git commit -asm.`
- 2 *hack, hack...* `$ git commit -asm'try this'`
- 3 *hack, hack...* `$ git commit -asm'try that'`
- 4 *rebuild... Ok it works, let's produce a clean patch:*
`$ git rebase -i [...]`

Some packages don't rebuild twice in a row properly, because their clean target is broken...

I laugh at those:

```
$ git clean -d && git reset --hard
```

The END!

If you're not too bored already, I'll gladly answer your questions now.

And remember, git has a tremendous community.

```
<mailto:git@vger.kernel.org>
```

Also consult the documentation:

```
/usr/share/doc/git-doc/index.html2.
```

²I assume you have git-doc installed, it will soon be part of base