

CPAN Patching with Git

by Yanick Champoux
yanick@babyl.dyndns.org



Git, the distributed version control system originally written by Linus Torvalds for the Linux kernel, has grown tremendously in popularity in the last few years. A big part of its success is the ridiculous ease with which one can turn any directory into an ad hoc, versioned repository. Indeed, all it takes is the magic incantation and voilà, instant versioning control:

```
$ git init
$ git add .
$ git commit -m "original commit"
```

With such a low overhead, Git is not only useful for code sharing, but is also a very attractive tool to easily add versioning, branching, or history capabilities to any project or task.

In this article, I'll show you how to use Git with to easily prepare a patch for a CPAN module.

■ Step 1: From CPAN into my grubby mi^D^D Git----

So, I've found a bug in a CPAN module and, infused with the proper dose of hubris or impatience for the task, decided to write a patch for it. Excellent!

But before going on a patching rampage, I must first set my baseline and download the module's latest release. To do so, I use a script called `git-cpan-init`. The script's functionality can be divided into three distinct parts: 1) retrieval of the module's distribution file from CPAN, 2) its extraction, and 3) gitification:

In *Code Listing 1 (next page)*, I use `CPANPLUS` to fetch the module (which I pass as an argument to the script at line 12) from CPAN. Although I don't use it in this part, I import `autodie` to take care of the calls to `unlink`, `mkdir`, `open` and such that will happen later in the script, so that I won't have to `die`

The second part of the job is straightforward; I use `Archive::Tar` to extract the distribution's files into the current directory and, once I'm done, I tidily remove the now-useless archive file.

The third part is where I bring Git into the picture. I create an empty repository structure (line 37) that I populate with everything in the current directory, including all sub-directories (lines 40 and 41). To put emphasis on the fact that the master branch's purpose is to track the CPAN versions of the module, I rename my master branch to `cpan` (line 42). Finally, I open the distribution's `META.yml` and, if I find the module's version inside, tag the current commit with it.

■ Module `Git.pm` not found? -----

If you don't have `Git.pm` on your system, download Git's source and look for the `perl` subdirectory. Inside, you'll find a familiar `Makefile.PL` file. Doing the usual dance (`perl Makefile.PL; make; make install`) will install the missing module.

■ Making `git-cpan-init` part of the family-----

Now that I have a snazzy script, I need to integrate it with Git. To do so, I find out where the `git-*` programs reside on my computer. Up to version 1.5 of Git, those programs should be either in `/usr/bin` or `/usr/local/bin`, for version 1.6, they should be `/usr/libexec/git-core` or `/usr/local/libexec/git-code`. I drop `git-cpan-init` in that directory and that's it. I'm done!

Because my script begins with `git-`, Git considers it to be just another member of its suite of help programs, and everything, including command completion if I have it enabled, will Just Work.

With `git-cpan-init` in place, I can now import the CPAN module I wish to patch:

```
$ mkdir L-133t
$ cd L-133t
$ git cpan-init Language::133t
downloaded distribution at Language-133t-
0.03.tar.gz
extracting distribution in current
directory
initializing Git repository

$ git status
# On branch cpan
nothing to commit (working directory clean)
```

■ Is there a repository in the room? -----

Maybe the latest released version of the module still ain't fresh enough for me. I could also see if the development code is available through a public repository. Nicely enough, the specification for `META.yml` has provisions for a repository resource field. This field is not yet widely used (nor, indeed, known), but it still doesn't hurt to look.

In `git-cpan-clone` in *Code Listing 2 (page 20)*, I play fast and loose and only retrieve the distribution's `META.yml` from CPAN. If I find that it does contain a repository resource, I try to figure out if it's a Git repository. This is more tricky than it seems, as the URL of a Git repository can use Git's own protocol

Download code at <http://www.theperlreview.com/Subscribers/CodeArchive>