Git tip: How to "merge" specific files from another branch

February 25, 2009

Problem statement

Part of your team is hard at work developing a new feature in another branch. They've been working on the branch for several days now, and they've been committing changes every hour or so. Something comes up, and you need to add *some* of the code from that branch back into your mainline development branch. (For this example, we'll assume mainline development occurs in the `master` branch.) You're not ready to merge the entire feature branch into `master` just yet. The code you need to grab is isolated to a handful of files, and those files don't yet exist in the `master` branch.

Buckets o' fail

This seems like it should be a simple enough task, so we start rummaging through our Git toolbox looking for just the right instrument.

Idea, the first. Isn't this exactly what `git cherry-pick` is made for? Not so fast. The team has made numerous commits to the files in question. `git cherry-pick` wants to merge a commit - not a file - from one branch into another branch. We don't want to have to track down all the commits related to these files. We just want to grab these files in their current state in the feature branch and drop them into the `master` branch. We *could* hunt down the *last* commit to each of these files and feed that information to `git cherry-pick`, but that still seems like more work than ought to be necessary.

Idea, the second. How 'bout `git merge --interactive`? Sorry. That's not actually

a thing. You're thinking of `git add --interactive` (which won't work for our purposes either). Nice try though.

Idea, the third. Maybe we can just merge the whole branch using `--squash`, keep the files we want, and throw away the rest. Um, yeah, that *would* work. Eventually. But we want to be done with this task in ten seconds, not ten *minutes*.

Idea, the fourth. When in doubt, pull out the brute force approach? Surely we can just check out the feature branch, copy the files we need to a directory outside the repo, checkout the `master` branch, and then paste the files back in place. Ouch! Yeah. Maybe, but I think we might have our Git license revoked if we resort to such a hack.

The simplest thing that could possibly work

As it turns out, we're trying too hard. Our good friend `git checkout` is the right tool for the job.

```
git checkout source_branch <paths>...
```

We can simply give `git checkout` the name of the feature branch [1] and the paths to the specific files that we want to add to our master branch.

```
$ git branch
* master
  twitter_integration
$ git checkout twitter_integration app/models/avatar.rb db/migrate/20090223104419_create_
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   app/models/avatar.rb
#       new file:   db/migrate/20090223104419_create_avatars.rb
#       new file:   test/functional/models/avatar_test.rb
#       new file:   test/unit/models/avatar_test.rb
#
```

```
$ git commit -m "'Merge' avatar code from 'twitter_integration' branch"
[master]: created 4d3e37b: "'Merge' avatar code from 'twitter_integration' branch"
4 files changed, 72 insertions(+), 0 deletions(-)
create mode 100644 app/models/avatar.rb
create mode 100644 db/migrate/20090223104419_create_avatars.rb
create mode 100644 test/functional/models/avatar_test.rb
create mode 100644 test/unit/models/avatar_test.rb
```

Boom. Roasted.

Notes

[1] `git checkout` actually accepts any tree-ish here. So you're not limited to grabbing code from the current tip of a branch; if needed, you can also check out files using a tag or the SHA for a past commit.

Jason Rudolph
Engineer at GitHub, runner, world traveler, and aspiring cookist

Other posts you might enjoy

10 reasons to upgrade your old Git installation

Defining Atom commands in your init script

As creative as we make time to be

Programming achievements: How to level up as a developer

Testing anti-patterns: Incidental coverage