The Mijingo Blog

Latest news, updates, free tutorials, and more from Mijingo.

Creating and Applying Patch Files in Git

by Ryan Irelan

In a previous article, I talked about how to use git-cherry-pick to pluck a commit out of a repository branch and apply it to another branch.

It's a very handy tool to grab just what you need without pulling in a bunch of changes you don't need or, more importantly, *don't want*.

This time the situation is the same. We have a commit we want to pull out of a branch and apply to a different branch. But our solution will be different.

Instead of using `git-cherry-pick` we will create a patch file containing the changes and then import it. Git will replay the commit and add the changes to the repository as a new commit.

What is git-fomat-patch?

`git-format-patch` exports the commits as patch files, which can then be applied to another branch or cloned repository. The patch files represent a single commit and Git replays that commit when you import the patch file.

`git-format-patch` is the first step in a short process to get changes from one copy of a repository to another. The old style process, when Git was used locally only without a remote repository, was to email the patches to each other. This is handy if you only need to get someone a single commit without the need to merge branches and the overhead that goes with that.

The other step you have to take is to import the patch. There are a couple options for that but we'll use the simplest one available.

Let's create our patch file.

Using git-format-patch

I am on the repository `the-commits`, which is the sample repository I used in my Git version control courses. I have the `experimental_features` branch checked out.

This `experimental_features` branch has an important change in it that I want to bring to a feature branch I have going. This feature branch is going to be merged into the development branch (and eventually the master branch) so I only want to include non-experimental changes. Because of that I don't want to do a merge because I'd like to not pull in the other features that are half-baked and would mess up my production-path branches.

Here's the latest when I run `git-log`:

```
 $ git log
commit 4c7d6765ed243b1dbb11d8ca9a28548561e1e2ef
Author: Ryan Irelan
Date:   Wed Aug 24 08:08:59 2016 -0500

another experimental change that I don't want to allow out of this branch

commit 1ecb5853f53ef0a75a633ffef6c67efdea3560c4
Author: Ryan Irelan
Date:   Mon Aug 22 12:25:10 2016 -0500

a nice change that i'd like to include on production

commit 4f33fb16f5155165e72b593a937c5482227d1041
Author: Ryan Irelan
Date:   Mon Aug 22 12:23:54 2016 -0500

really messed up the content and markup and you really don't want to apply this com

commit e7d90143d157c2d672276a75fd2b87e9172bd135
Author: Ryan Irelan
Date:   Mon Aug 22 12:21:33 2016 -0500

rolled out new alpha feature to test how comments work
```


Subscribe to the Blog

Subscribe via RSS
Subscribe via Email

Browse the Blog

Web Performance Testing
Performance Budget
Performance Testing
Behind the Scenes
Classroom Training
Command Line
Community Resources
Craft CMS
Twig
ExpressionEngine
Free Tutorials
Git
iPhone App Design
JavaScript Task Runners
Learning Techniques
Localhosting
Mijingo Courses
Mijingo News
OS X
Research & Experiments
SVG

Learn with Mijingo

Up and Running with Craft 3
Up and Running with ExpressionEngine
Git Extras
Control Flow in Twig
Fundamentals of Craft Commerce
Flexible Twig Templates in Craft
Up and Running with Craft
Up and Running with Flexbox
Git: Under the Hood
Web Performance Testing
Craft Plugin Development
Twig Templates in Craft
Git: The Next Steps
OS X Shell Tricks
Command Line Fundamentals
Up and Running with SVG
Reliable Localhosting
JavaScript Task Runners: Grunt & Gulp
Static Websites with Jekyll
Learning ExpressionEngine
Deploying Websites
Basics of Grids
Up and Running with Sass
Fundamentals of CSS3
Fundamentals of HTML5
Responsive Web Design

The commit with the hash `1ecb5853f53ef0a75a633ffef6c67efdea3560c4` is the one I'd like to pull into my feature branch via a patch file.

We do that using the command `git-format-patch`. Here's the command:

```
$ git format-patch a_big_feature_branch -o patches
```

We pass in the branch with which we want Git to compare against to create the patch files. Any commits that are in our current branch (`experimental_features`) but not in the `a_big_feature_branch` will be exported as patch files. One patch file per commit. We used the `-o` flag to specify the directory where we want those patches saved. If we leave that off, Git will save them to the current working directory.

When we run it we get this:

```
$ git format-patch a_big_feature_branch
patches/0001-rolled-out-new-alpha-feature-to-test-how-comments-wo.patch
patches/0002-really-messed-up-the-content-and-markup-and-you-real.patch
patches/0003-a-nice-change-that-i-d-like-to-include-on-production.patch
patches/0004-another-experimental-change-that-I-don-t-want-to-all.patch
```

Those four patch files (named sequentially and with a hyphenated version of the commit message excerpt) are the commits that are in the current branch but not the `a_big_feature_branch`.

Let's look at the guts of one of them.

```
From 4c7d6765ed243b1dbb11d8ca9a28548561e1e2ef Mon Sep 17 00:00:00 2001
From: Ryan Irelan
Date: Wed, 24 Aug 2016 08:08:59 -0500
Subject: [PATCH 4/4] another experimental change that I don't want to allow out of t

---
 index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/index.html b/index.html
index f92d848..46e4eb2 100644
--- a/index.html
+++ b/index.html
@@ -9,7 +9,7 @@
    <!-- Set the viewport width to device width for mobile -->
    <meta name="viewport" content="width=device-width" />

 <title>Little Git &amp; The Commits</title>
 <title>Little Git &amp; The Commits FEATURING ELVIS BACK FROM THE DEAD</title>

    <!-- Included CSS Files (Uncompressed) -->
--
2.7.4 (Apple Git-66)
```

It looks like an email, doesn't it? That is because all patch files are formatted to look like the UNIX mailbox format. The body of the email is the diff that shows which files have changed (in our case just `index.html`) and what those changes are. Using this file, Git will recreate the commit in our other branch.

Specifying a Single Commit

In this situation, I don't need all of those patch files. All but one are commits I *don't* want in my target branch. Let's improve the `git-format-patch` command so it only creates a patch for the one commit we do want to apply.

Looking back at the log, I know that the commit I want to apply has the hash of `1ecb5853f53ef0a75a633ffef6c67efdea3560c4`. We include that hash as an argument in the command, but precede it with a `-1` so Git only formats the commit we specify (instead of the entire history since that commit).

```
$ git format-patch a_big_feature_branch -1 1ecb5853f53ef0a75a633ffef6c67efdea3560c4
 outgoing/0001-a-nice-change-that-i-d-like-to-include-on-production.patch
```

Now we get a single patch file, which is much safer because there's no change we'll accidentally apply patches of changes we don't want!

We have the patch file, now how do we apply it to our branch?Using `git-am`.

What is `git-am`?

`git-am` is a command that allows you to apply patches to the current branch. The `am` stands for "apply (from a) mailbox" because it was created to apply emailed patches. The handy thing about `git-am` is that it applies the patch as a commit so we don't have to do anything after running the command (no `git-add`, `git-commit` etc.).

The name `git-am` is a little strange in the context of how we're using it but

Up and Running with Git
Building an ExpressionEngine Add-on
PIP and virtualenv
Python for PHP Developers
Budgets with Soulver
MySQL and ExpressionEngine
Securing ExpressionEngine 2
ExpressionEngine 2: A Quick-Start Guide

fear not: the result is exactly what we want.

Let's apply a patch and see how it works.

Using `git-am`

The first thing we need to is switch over to our target branch. For this example we'll move to the branch we compared against in the `git-format-patch` command.

```
$ git checkout a_big_feature_branch
```

After that we're ready to apply the patch file with the commit we want to include.

Note: I'm working in the same repository on the same computer. When I switch branches, the patch file comes with me because it is still an untracked file. If I staged and committed the patch file then I'd need to find another way to make it accessible. You could do this by moving the patch file out of your repository to where you can access it when on the destination branch.

Because we refined the `git-format-patch` we only have one patch file in the patches directory:

```
patches/0001-a-nice-change-that-i-d-like-to-include-on-production.patch
```

To apply the patch to the current branch, we use `git-am` and pass in the name of the patch we want to apply.

```
$ git am patches/0001-a-nice-change-that-i-d-like-to-include-on-production.patch
```

And the we get confirmation that the patch was successfully applied:

```
Applying: a nice change that i'd like to include on production
```

Looking at the log now we see our change is replayed as a commit in the current branch:

```
 $ git log
commit 69bb7eb757b2356e365934fdbea744877c3092bb
Author: Ryan Irelan
Date:   Mon Aug 22 12:25:10 2016 -0500

a nice change that i'd like to include on production
```

And now our change is there!

Note that the new commit has a different hash because it is part of a different working tree than the one we formatted as a patch.

Learning More About Git

To learn more about Git check out our Git courses, lessons and tutorials.

LEARN MORE ABOUT GIT

Filed Under: *Git*