How to duplicate a file while preserving git line history

devblogs.microsoft.com/oldnewthing/20190919-00

September 19, 2019

Raymond Chen

Today, we're going to duplicate a file while preserving git line history.

This could be useful if you want two copies of a component, say, one where you are doing a bunch of disruptive work, and another that remains largely unchanged. The project continues to use the old, stable version, but there's a feature flag to switch to the new, exciting one. Eventually, you'll make the new, exciting one the default version.

When you do this, you want the line history of the new version to be the same as the line history of the old version, because the new version is basically a fork of the old version.

Again, let's use the same scratch repo as we did for the last few days. You can follow the same copy/paste script, or you can take your existing scratch repo and `git reset --hard ready` to get it back into its "ready to start experimenting" state.

Let's set up a scratch repo to demonstrate. I've omitted the command prompts so you can copy-paste this into your shell of choice and play along at home. (The timestamps and commit hashes will naturally be different.)

```
git init

>foods echo apple
git add foods
git commit --author="Alice <alice>" -m created

>>foods echo orange
git commit --author="Bob <bob>"    -am orange

git blame foods

^62ef37c (Alice 2019-09-19 07:00:00 -0700 1) apple
335acb1b (Bob   2019-09-19 07:00:01 -0700 2) orange
```

We employ our standard trick: Create a branch where the desired new file appears to have been created via a rename of the original file. And then restore the original file.

```
git checkout -b dup

git mv foods foods-new
git commit --author="Greg <greg>" -m "duplicate foods to foods-new"

git checkout HEAD~ foods
git commit --author="Greg <greg>" -m "restore foods"

git checkout -
```

On this branch, we renamed `foods` to `foods-new`. When git traces the history of the `foods-new` file, it'll see that the file was created via rename from `foods`, so git will use `food`'s history to build the line history.

And then we bring back the original `foods` file. We use the `git checkout HEAD~ foods` command to restore the file from a specific commit, namely the commit before we renamed it away.

```
git merge --no-ff dup

Merge made by the 'recursive' strategy.
 foods-new | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 foods-new
```

The `dup` branch deleted the `foods` file, and then restored it. That means there was no net change to the file in the `dup` branch, and even `git log` won't notice it by default. If you do a log of the `foods` file, the merge doesn't even show up.

```
git log --oneline foods

                ← the merge doesn't appear
335acb1 orange
62ef37c created
```

The line histories of the two files are identical, because the `foods-new` was created at the same time an identical `foods` file disappeared, which made git consider the operation to be a rename for the purpose of history tracking.

```
git blame foods

^62ef37c (Alice 2019-09-19 07:00:00 -0700 1) apple
335acb1b (Bob   2019-09-19 07:00:01 -0700 2) orange

git blame foods-new

^62ef37c foods (Alice 2019-09-19 07:00:00 -0700 1) apple
335acb1b foods (Bob   2019-09-19 07:00:01 -0700 2) orange
```

Raymond Chen

Follow