Why perform all these complex git machinations when you can just tweak the command line to get what you want?

devblogs.microsoft.com/oldnewthing/20190920-00

September 20, 2019

Raymond Chen

The recent series on splitting out files (and the earlier series on combining files) went through complex git machinations to get the desired effect. Are we just doing things the hard way? Just make a commit that contains the desired changes all in one go, no fuss, no muss, no bother. By fiddling with command line options, say by using `-C20%` or `-C -C -C`, we can get `git blame` to produce the desired results.

But that's not practical for various reasons.

First of all, everybody in the team[1] has to remember to use the correct command line switches. The people over in QA who field the bug are probably not going to remember to use Greg's special command line options when running `git blame` on this specific file. They're just going to use the default options, and that means that skipping the fancy git commit tricks and relying on special command line options means that all of the tickets will end up assigned to Greg.

Greg could try to correct every person who blames him for a line of code by saying, "No, I'm not responsible for that line of code. If you run `git blame` with this special command line, you'll see that it's not really me. It was Bob!"

This solution makes nobody happy. Greg is frustrated at having to explain this over and over again. And all of Greg's teammates are frustrated that every time they assign a ticket to Greg, there's a good chance Greg is going to get all annoyed with them and say, "But if you run `git blame` in this *very specific way*, then you can see that I'm not the one to blame."

From the point of view of everybody who isn't Greg, it looks like Greg is just shirking his responsibility and trying to shift blame. He found a very special command line that causes his name to disappear from `git blame`, and he insists that everyone use it. "Greg probably spent days trying out different command line options until he found the magic one that causes his name to disappear."

Furthermore, Greg's preferred command line switches may conflict with somebody else's preferred command line switches. Greg eventually strong-arms everybody into running `git blame` with the `-C20%` switch. A few weeks later, Hannah does another split-out and says, "Oh, in order for this to work, you need to pass `-C15%`." But passing `-C15%` causes Isaac's change from two weeks ago to start producing false-positives and treat a file as having been copied, even though it wasn't.

Basically, this is another case of using a global solution to a local problem. The local problem is "This brief sequence of commits needs special treatment in order to produce the results I want." The command line is a shared resource, and for each command, there is only one. If different sequences of commits require different settings, then you'll never find a setting that works for everyone.

What's more, the preferred command line switches may not be suitable for your repo. For example, the Windows repo has 3 million files, and it is not unusual to see tens of thousands of files modified by a single commit. Any nonzero value for `-C` or `-M` would cause git to perform millions of file-to-file comparisons. Instead, you'll probably hit the `renameLimit`, and the `-C` and `-M` options will be ignored entirely. When your change merges into another branch as part of a large payload, no special command line will save you.

Finally, you may not have control over the command line switches at all. When you view the `git log` or `git blame` on a Web-hosted repo, the server decides what command line options to use, not you. (Maybe there's an administrative setting to control the options used by the Web server, but then you're back to the "global solution" problem.)

The techniques I've been using will get the desired results independent of the command line options. Even in the face of `-M0 -C0`, the special commit sequences will still work because they provide perfect hash matches, which means that git will detect the rename without having to do any file-by-file comparison.

In other words, these special commit sequences are, in a sense, universal: They will work on any repo, no matter how it is configured.

[1] And there is definitely a team here, because if there is only one developer working in the repo, then there would be no need to do fancy blame games. Everything in the repo is blamed on one person!

Raymond Chen

Follow