How can I bulk-revert a subdirectory of a repo to an earlier commit?

devblogs.microsoft.com/oldnewthing/20201005-00

October 5, 2020

Raymond Chen

Last time, we looked at ways to return a repo to a state that matched an earlier commit. But what if you don't want to return the entire repo to that state, just a subdirectory?

A simple way to return a file or group of files to an earlier state is to check them out based on an earlier commit:

```
# Warning: Read caveats below before using
git checkout A -- some_file
git checkout A -- wildcards*
git checkout A -- some_directory
git checkout A -- .
```

This takes file or group of files as it was as of a commit and copies them to the index as well as putting them in your working tree.

The caveat is that it affects only files that were present in commit A. If a file was added some time after commit A, then the `git checkout` won't delete it.[1]

Fortunately, you can detect whether this has happened by doing a `diff`:

```
git diff --cached A --name-only -- some_file
git diff --cached A --name-only -- wildcards*
git diff --cached A --name-only -- some_directory
git diff --cached A --name-only -- .
```

If the output is empty, then everything matches.

If the output is not empty, then *something* is different, but the output doesn't tell the whole truth. If any files were renamed, then the output will show the newly-created file that resulted from the rename, but will omit the no-longer-present file. To get the true list of affected files, you need to disable rename detection.

```
git diff --cached A --name-only --no-renames -- whatever
```

Even though the `git checkout` method has its flaws, they are know flaws that you can detect with some follow-up commands, and `git checkout` does have the advantage of using commands that you are already familiar with.

But my preference when trying to do tree surgery is just to do tree surgery directly, rather than trying to find equivalent commands that have the same effect as tree surgery.

```
git rm --cached -r subdirectory
git read-tree --prefix:subdirectory A:subdirectory
```

The first command removes the subdirectory from the index. This is a preparatory step, because `git read-tree` will get mad if you try to graft a tree into a place where a tree already exists.

The second command is the money. It takes a subtree from commit A and grafts into into the index at the specified location.

In the above example, I placed the subtree in the same location as it came from, but you don't have to do it that way.

```
git read-tree --prefix:archive/src A:src
```

The above version takes the `src` subtree from commit A and adds it to the index as a subdirectory named `archive/src`.

Once the changes are in the index, you can commit them directly with `git commit`, or you can `git reset` them to unstage the changes, or you can `git add` more files to the change (say, to update a `README`). If you want to make changes to the files you staged, you can copy them into your worktree with `git checkout --`.

Basically, the grafting is all staged, and you can use regular git commands to do whatever it is you want to do next.

[1] That doesn't stop people from using it for this purpose. I hope they understand where the holes are.

Raymond Chen

Follow