git commit-tree parlor tricks, Part 9: How can I bulk-revert an entire repo to an earlier commit?

devblogs.microsoft.com/oldnewthing/20200928-00

September 28, 2020

Raymond Chen

Suppose you've made a bunch of commits to a branch, and then you decide that you want to roll back the entire repo to an earlier commit. Just pretend the last dozen commits never happened. The branch policy prevents force-pushes, so you will have to make a new commit that effectively reverts a large number of commits.

A M1 M2

We start with some commit A, and there have been some commits M1 and M2 on top of it. What's the easiest way to do a bulk revert back to A?

You might think you could merge commit A with the `-s theirs` option, but that doesn't work because commit A is already in the history of the branch, so the merge does nothing.

Another thing you could try is to `git checkout A -- .` from the root of the repo, saying that you want to take every single file from commit `t` and put it into the current tree. This mostly works, except that any files added after commit A will not be deleted. The `git checkout A -- .` will update all the files that were present in A, but any files added in M1 and M2 will not be deleted.

My old standby is `git commit-tree`. In this case, we want to create a commit on top of `HEAD` with the contents of an earlier commit.

```
git commit-tree A^{tree} -p HEAD -m "Bulk revert back to A"
```

Note: If using the Windows `cmd` command prompt, you need to type

```
git commit-tree A^^{tree} -p HEAD -m "Bulk revert back to A"
```

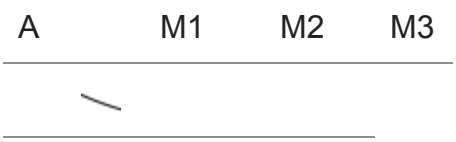
for reasons discussed earlier.

The `git commit-tree` command will spit out a hash, which you can fast-forward to.

If you want to express this as a merge, you could say

```
git commit-tree A^{tree} -p HEAD -p A -m "Bulk revert back to A"
```

This generates a merge commit that would not normally be found in nature: Merging a commit that you already have.



We have manufactured a commit M3 which represents a merge of its own ancestor commit A.[1]

Now, there are plenty of other ways to accomplish the same thing. I like `commit-tree` because it directly creates exactly the commit I want, and it does so without affecting the index or working tree.

On the other hand, it doesn't give you a chance to inspect and possibly alter the result before committing. Then again, maybe that's not a problem. After you fast-forward to the manually-created commit, you can make whatever additional changes you like and either make a new commit on top, or amend the previous commit.

But maybe you prefer to have the commit staged. You can do that by reading the desired target into the index and asking for the working tree to be updated to match. Make sure your index and working tree are clean by doing a `git status` and verifying that it says "Nothing to commit, working tree clean." And then do this:

```
git read-tree -mu A
```

This makes the index and working tree match the tree from commit A. (If your working tree and index are not clean before doing this, the results will be merged in, which will probably be a mess.) You can now inspect the results, make additional changes, whatever, before you commit.

If you want to commit this as a "not found in nature" merge, you'll still have to do some `git commit-tree` magic:

```
git write-tree
```

This will print a tree hash. Feed that hash to the next line:

```
git commit-tree ⟨hash⟩ -p HEAD -p A -m "Bulk revert back to A"
```

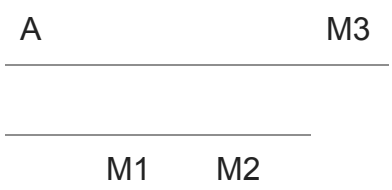
This will print a commit hash. Fast-forward to that commit to complete the artificial merge.

Okay, this is all great, but what if you didn't want to revert the entire repo, but just a part of it? We'll look at that next week.

Bonus chatter: If you wanted to express this as a pure revert rather than a merge, then omit the `-p A` from the command lines. But using the merge has the nice side effect of assigning `git blame` to the commits that led to A. The commits M1 and M2 will never be assigned blame, which makes sense, since all their changes were reverted.

That nice side effect on `git blame` means that this technique is useful if you are reverting the last change, since it takes the change and its revert out of the `git blame`.

[1] Note that the topologically equivalent diagram does occur in nature:



This represents the situation where a no-fast-forward merge is taken from a topic branch.

Raymond Chen

Follow