Mundane git commit-tree tricks, Part 4: Changing a squash to a merge

devblogs.microsoft.com/oldnewthing/20190509-00

May 9, 2019

Raymond Chen

Suppose you performed a `git merge --squash`, and then later realized that you meant it to be a true merge rather than a squash merge.

In diagrams, suppose you have this:

A	M1		M2	master
	F1	F2		feature

From a starting commit A, the master branch continues with a commit M1. From that same starting commit A, a feature branch adds commits F1 and F2. Now you decide to take the feature branch into the master branch, and you resolve the merge as a squash. This means that the resulting commit M2 technically has only a single parent M1. The other alleged parent F2 is just a dotted line, indicating that it is just a figment of your imagination.

You then realize that you should have accepted the feature branch as a true merge rather than a squash. What can you do?

Naturally, you could hard reset the master branch back to M1 and redo the merge. But that means you have to redo all the merge conflicts, and that may have been quite an ordeal. And if that was a large merge, then even in the absence of conflicts, you still have a lot of files being churned in your working directory.

Much faster is simply to create the commit you want and reset to it.

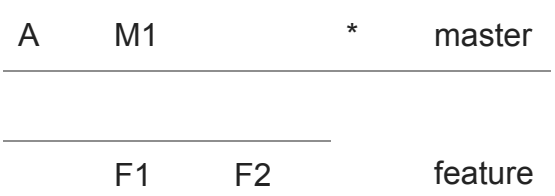
```
git commit-tree HEAD^{tree} -p HEAD~ -p F2 -m comment
```

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

```
git commit-tree HEAD^^{tree} -p HEAD~ -p F2 -m comment
```

for reasons discussed earlier.

This creates a tree identical to what is currently checked in (which is the merge result), whose parents are M1 and F2. In other words, it colors in the dotted line.



The result of the `git commit-tree` command is a hash printed to stdout. You can `git reset` to that hash to make that the branch head.

Raymond Chen

Follow