Ritesh Goyal: Git Rebase

Let's look at a branch rebase. Consider that we have two branches—*master* and *feature* —with the chain of commits shown in Figure 1 below. *Master* has the chain `C4->C2->C1->C0` and *feature* has the chain `C5->C3->C2->C1->C0`.

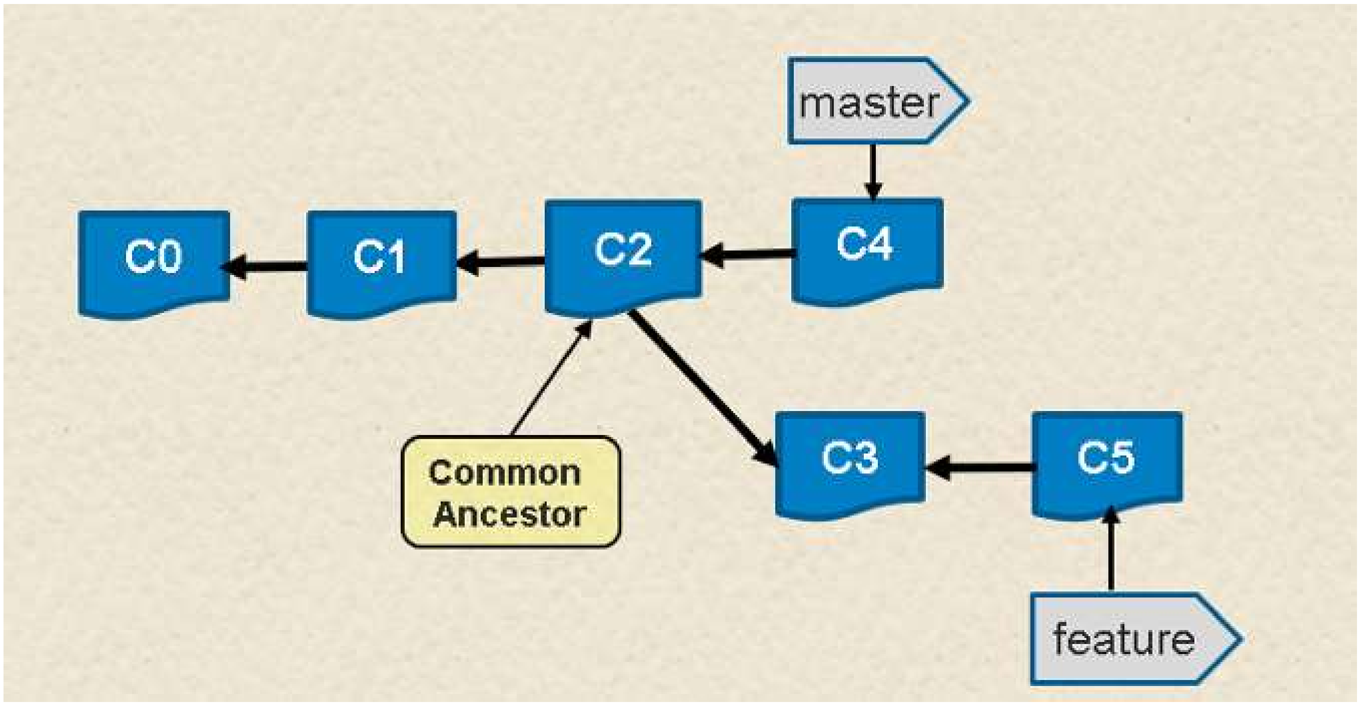

Fig. 1: Chain of commits for branches master and feature

If we look at the log of commits in the branches, they might look like the following. (The `C` designators for the commit messages are used to make this easier to understand.)

```
$ git log --oneline master
6a92e7a C4
259bf36 C2
f33ae68 C1
5043e79 C0

$ git log --oneline feature
79768b8 C5
000f9ae C3
259bf36 C2
```

```
f33ae68 C1
5043e79 C0
```

I tell people to think of a rebase as a "merge with history" in Git. Essentially what Git does is take each different commit in one branch and attempt to "replay" the differences onto the other branch.

So, we can rebase a feature onto *master* to pick up `C4` (e.g., insert it into feature's chain). Using the basic Git commands, it might look like this:

```
$ git checkout feature
$ git rebase master

First, rewinding head to replay your work on top of it...
Applying: C3
Applying: C5
```

Afterward, our chain of commits would look like Figure 2.

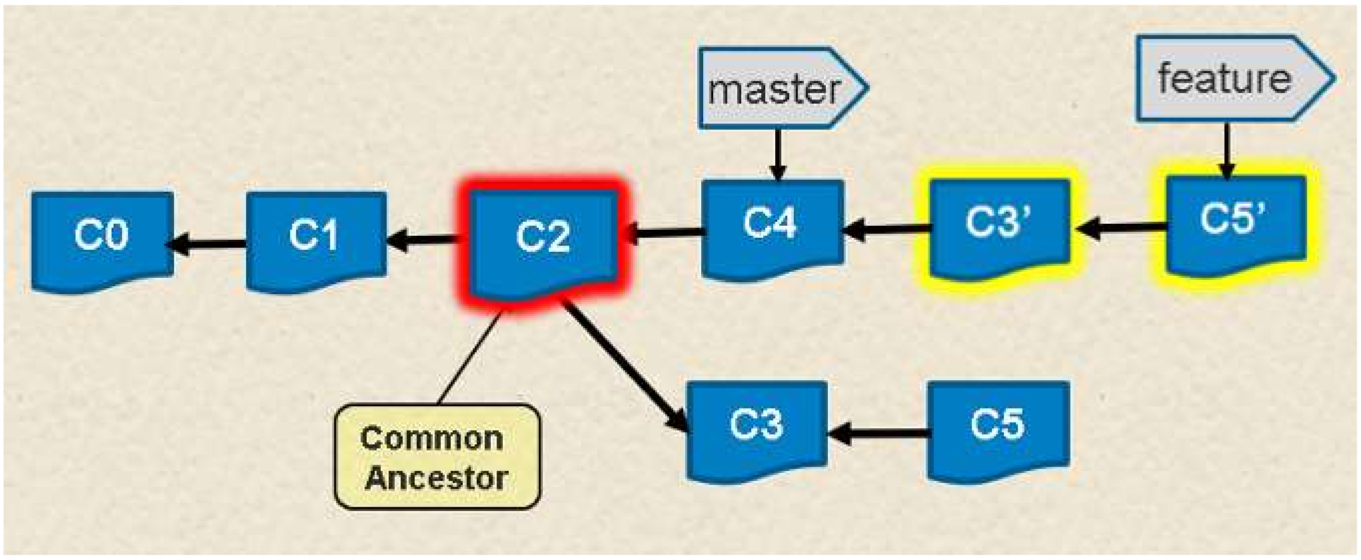

Fig. 2: Chain of commits after the `rebase` command

Again, looking at the log of commits, we can see the changes.

```
$ git log --oneline master
6a92e7a C4
259bf36 C2
f33ae68 C1
5043e79 C0

$ git log --oneline feature
c4533a5 C5
64f2047 C3
6a92e7a C4
259bf36 C2
f33ae68 C1
5043e79 C0
```

Notice that we have `C3'` and `C5'`—new commits created as a result of making the changes from the originals "on top of" the existing chain in *master*. But also notice that the "original" `C3` and `C5` are still there—they just don't have a branch pointing to them anymore.

If we did this rebase, then decided we didn't like the results and wanted to undo it, it would be as simple as:

```
$ git reset 79768b8
```

With this simple change, our branch would now point back to the same set of commits as before the `rebase` operation—effectively undoing it (Figure 3).

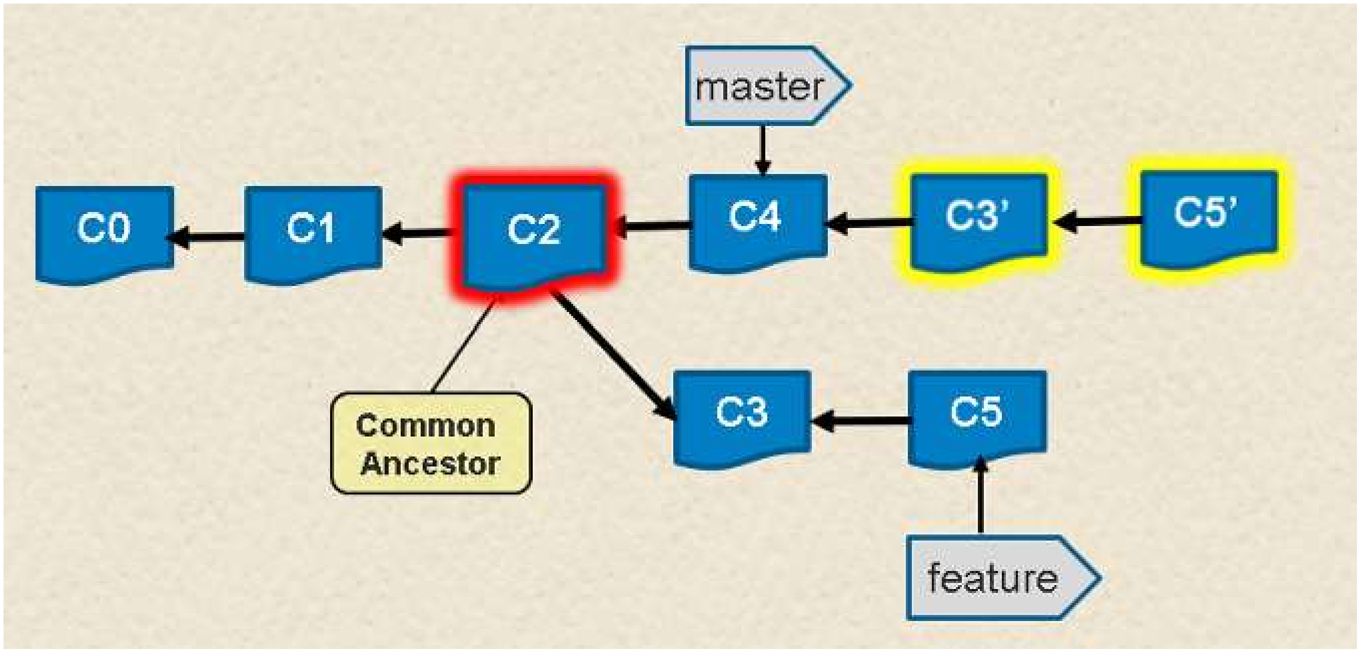

Fig. 3: After undoing the `rebase` operation

What happens if you can't recall what commit a branch pointed to before an operation? Fortunately, Git again helps us out. For most operations that modify pointers in this way, Git remembers the original commit for you. In fact, it stores it in a special reference named `ORIG_HEAD` within the `.git` repository directory. That path is a file containing the most recent reference before it was modified. If we `cat` the file, we can see its contents.

```
$ cat .git/ORIG_HEAD
79768b891f47ce06f13456a7e222536ee47ad2fe
```

We could use the `reset` command, as before, to point back to the original chain. Then the log would show this:

```
$ git log --oneline feature
79768b8 C5
000f9ae C3
259bf36 C2
f33ae68 C1
5043e79 C0
```