Handling Merge Conflicts

Explore how you can handle merge conflicts.

We'll cover the following

Don’t panic

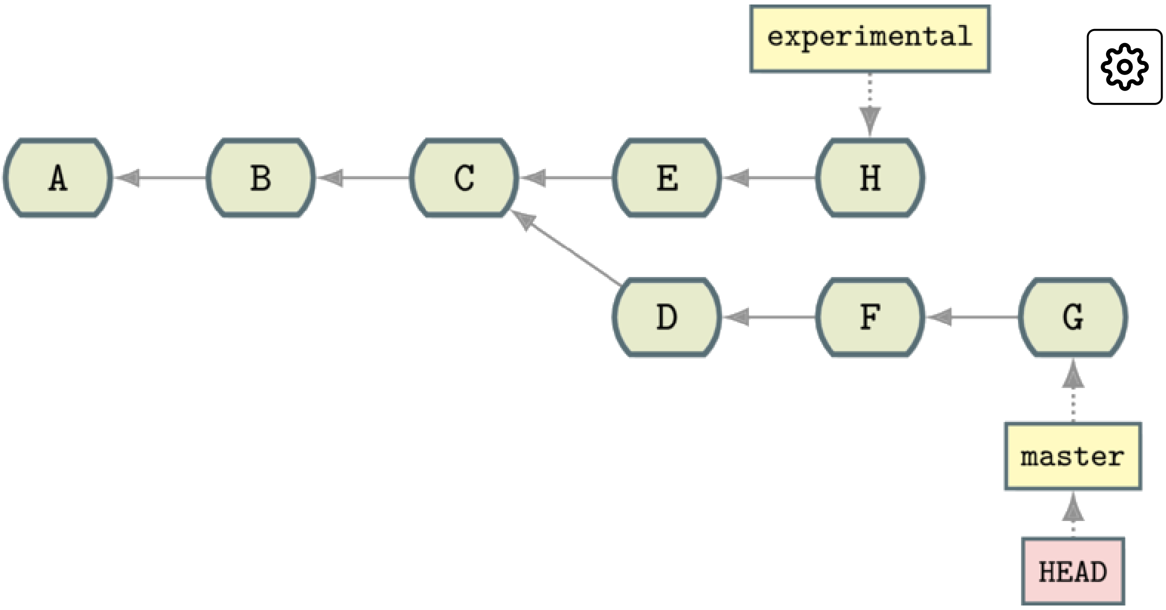
This is where a lot of people panic when merging, and it helps to understand what’s going on by running through it by hand and with a toy example. So, what exactly happens when you perform a merge?

Merge conflict

When you run a merge, Git looks at the branch you are on (here it is `master`) and the branch you are merging in (`experimental`) and works out the first common ancestor. In this case, it’s point `C`, as that’s where you branched `experimental`.

It then takes the changes on the branch that you are merging in from that *first common ancestor* and applies them to the branch you are on in one go. These changes create a new commit, and the `git log` graph shows the branches joined back up.

Sometimes the changes made on the branches conflict with one another. That means the changes altered the same lines. In this case, the `D`, `F`, and `G` of the `master` changed the same lines as the `E` and `H` of `experimental`.



Git doesn't know what to do with these lines. Instead of putting the `D`, `F`, and `G` in, should it put the `E` and `H` in? Or should it put them all in? If it should put them all in, then what order should they go in?

Changing lines around the same area in code can have disastrous effects. By default, Git does not make a decision when this happens. Instead, it tells you that there was a conflict and asks you to "fix conflicts and then commit the result".

If you look at `file1` now:

```
cat file1
```

Terminal 1

Terminal

All the lines from both branches are in the file.

There are three sections here:

1. The file up to line `C` is untouched, as there was no conflict.
2. Then we see a line with arrows indicating the start of a conflicting section. It is followed by the point in the repository that those changes were made on (in this case, `HEAD`).

3. Then a line of seven equals signs (`=======`) indicates the end of a conflicting set of changes. It is followed by the changes on the other conflicting branch (the `E` and `H` on the `experimental` branch).

Resolving merge conflicts

What you choose to do here is up to you as maintainers of this repository. You could add or remove lines as you wish until you are happy the merge has been completed, or replace it with something completely different. Git doesn't care.

When you are satisfied, you can commit your change, and the merge has been completed.

You could even leave the file as is (including the `=======` and `<<<<<<<` lines). But this is unlikely to be what you want! It's surprising how easily you can forget to resolve all the conflicting sections in your codebase when doing a merge.

When you are done, you can commit the change, and view the history with the `git log` command.

```
git commit -am 'merged experimental in'
git log --all --oneline --graph --decorate
```

Terminal 1

Terminal

Reading this from bottom to top, you can read commit `C` and commit `H` as being merged into the `HEAD` of `master`.

Git Log Diagrams

`git log` prefers to show the history from most recent to oldest, which is the opposite of the diagrams in this course (left-to-right). The `git man` pages also like to show time from left-to-right like this:

```
              A'--B'--C' topic
             /
D---E---F---G master
```

If you think this difference is confusing, I won’t disagree.

However, for `git log` it makes some sense. If you are looking at a repository with a long history, you are more likely to be interested in recent changes than older ones.

← Back

Merging Step by Step

Next →

Challenge: Merging

☑ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/handling-merge-conflicts__merging__learn-git-the-hard-way)