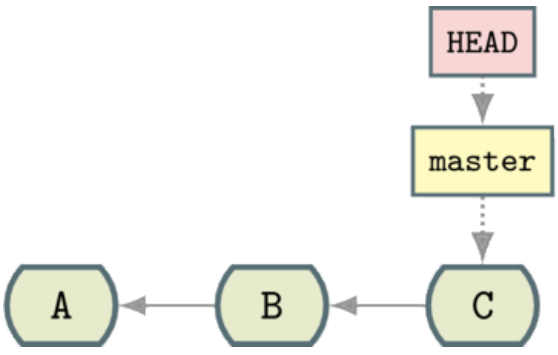
Outline of a Simple Rebase

To understand rebase, examine a simple visual example of it.

We'll cover the following

Commits on `master` branch

Let's say you have a set of changes on a `master` branch:



Simple Git repository with one main line

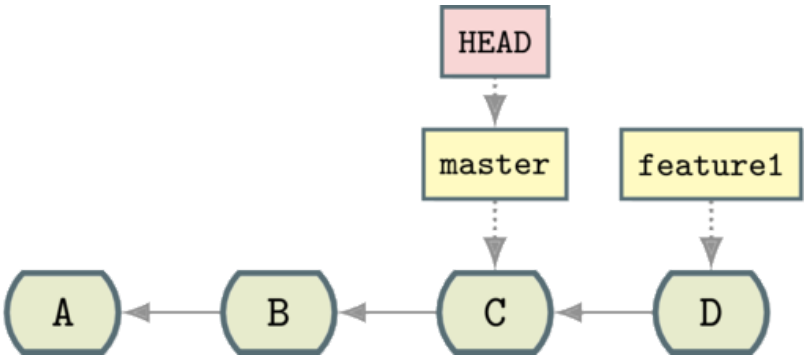
Reminder: Git Log Diagrams

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

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

Branching off to `feature1`

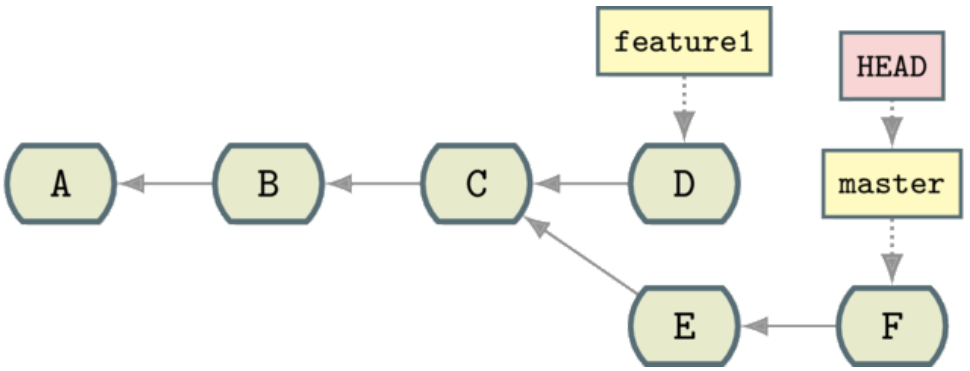
And at this point, you branch off to `feature1` and make another change:



feature1 branched off

Checking out `master` branch

Now, go back to `master` and make a couple more changes:



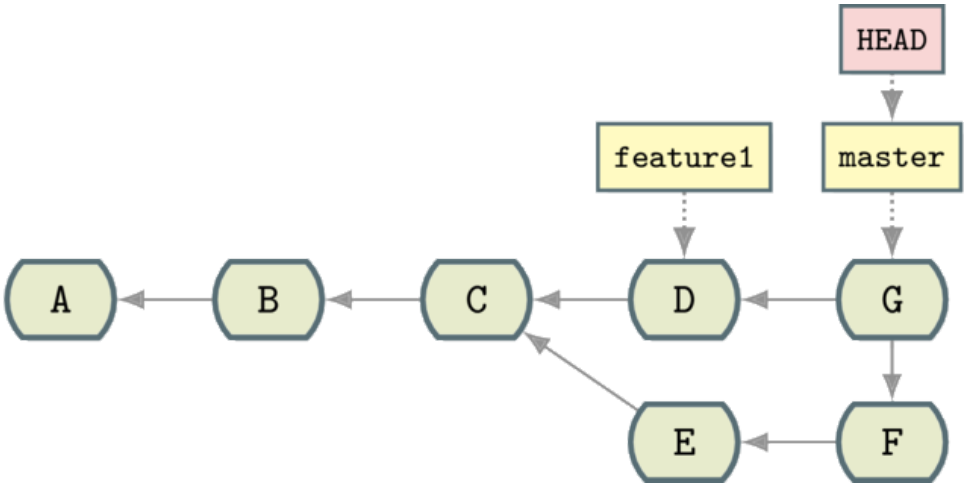
Two changes made on master

Think about this from the point-of-view of the developer of `feature1`.

They have made a change from point `C` on the `master` branch, but the situation there has moved on since.

Possibility of a merger

Now, if `master` wants to merge in the change on `feature1`, then it could merge it in and the tree would look like this:



feature1 and master branches merged in to "G"

Drawbacks of merging

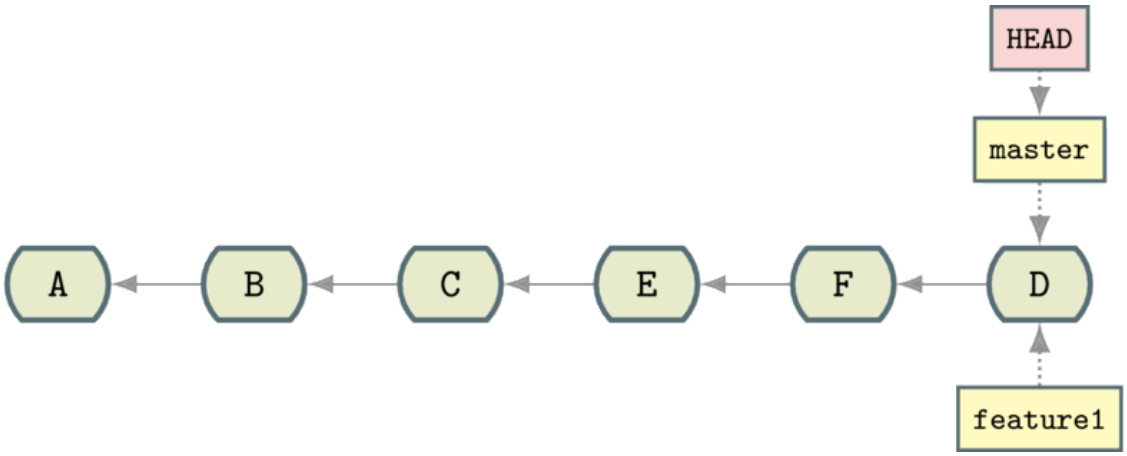
That is okay but not entirely desirable for two reasons:

- The history has just got more complicated (more commits).

- You have introduced an extra new change (G), which is the merge of D and F.

Another possibility?

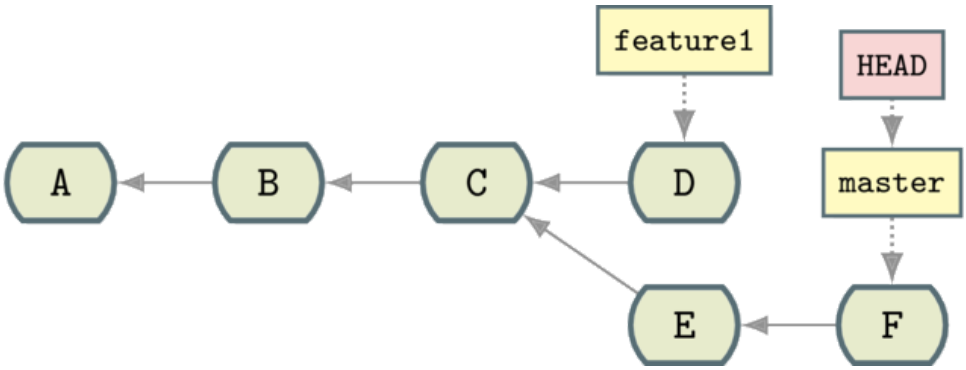
Wouldn't it be better if the history looked like this?



Proposed simpler history

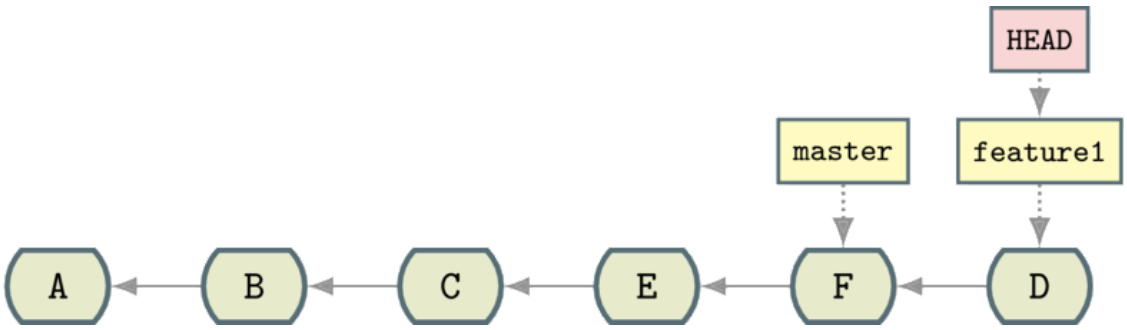
This is much cleaner and easier to follow. For example, if a bug was introduced in `D`, it's easier to find (e.g., using `git bisect`, which you will learn about soon). Also, the `feature1` branch can be safely deleted without any significant information being lost, making the history tidier and simpler.

If you remind yourself of the pre-merge situation (above), then you can visualize "picking up" the changes on the `feature1` branch and moving them to the `HEAD`.



Git rebase: before

To this:



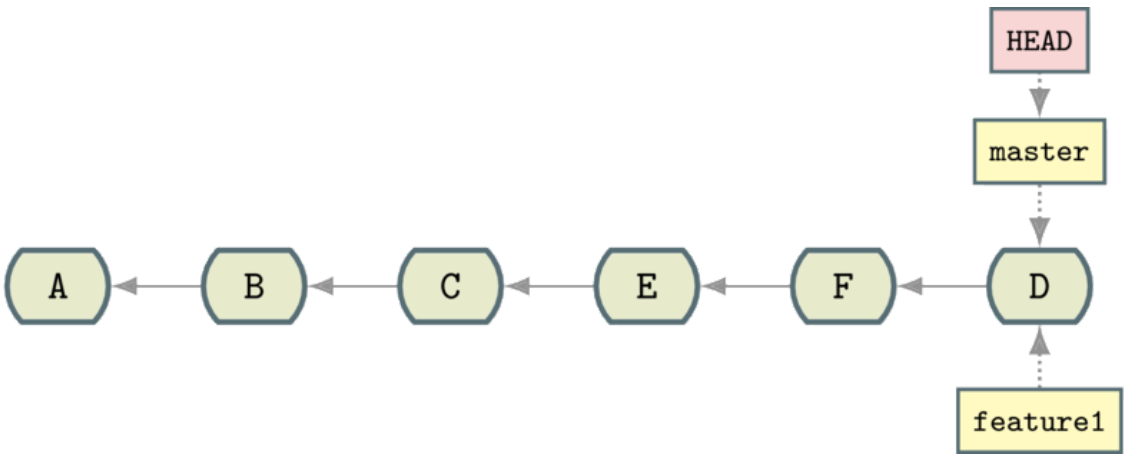
Git rebase: after

This is all what a simple rebase is.

What is a rebase?

A rebase is actually a more abstract concept that we will be covered in slightly more detail in a later section. But for now, you don't need to worry about that. In 99% of daily discussions about rebases, this is what people mean.

Once the above is done, you'll move `master` so it's pointing at the same place as `feature1`:



Final rebased state

Take a set of changes from a particular point and apply them from a different point: literally rebase your changes!

Squashing

Be aware that people also talk about rebasing to “squash” commits. This is a slightly different scenario that uses the same rebase command in a slightly different way. We will cover this part later.

Back

Introduction: Git Rebase

Next

Walkthrough of a Simple Rebase

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/outline-of-a-simple-rebase__git-rebase__learn-git-the-hard-way)