Walkthrough of a Simple Rebase

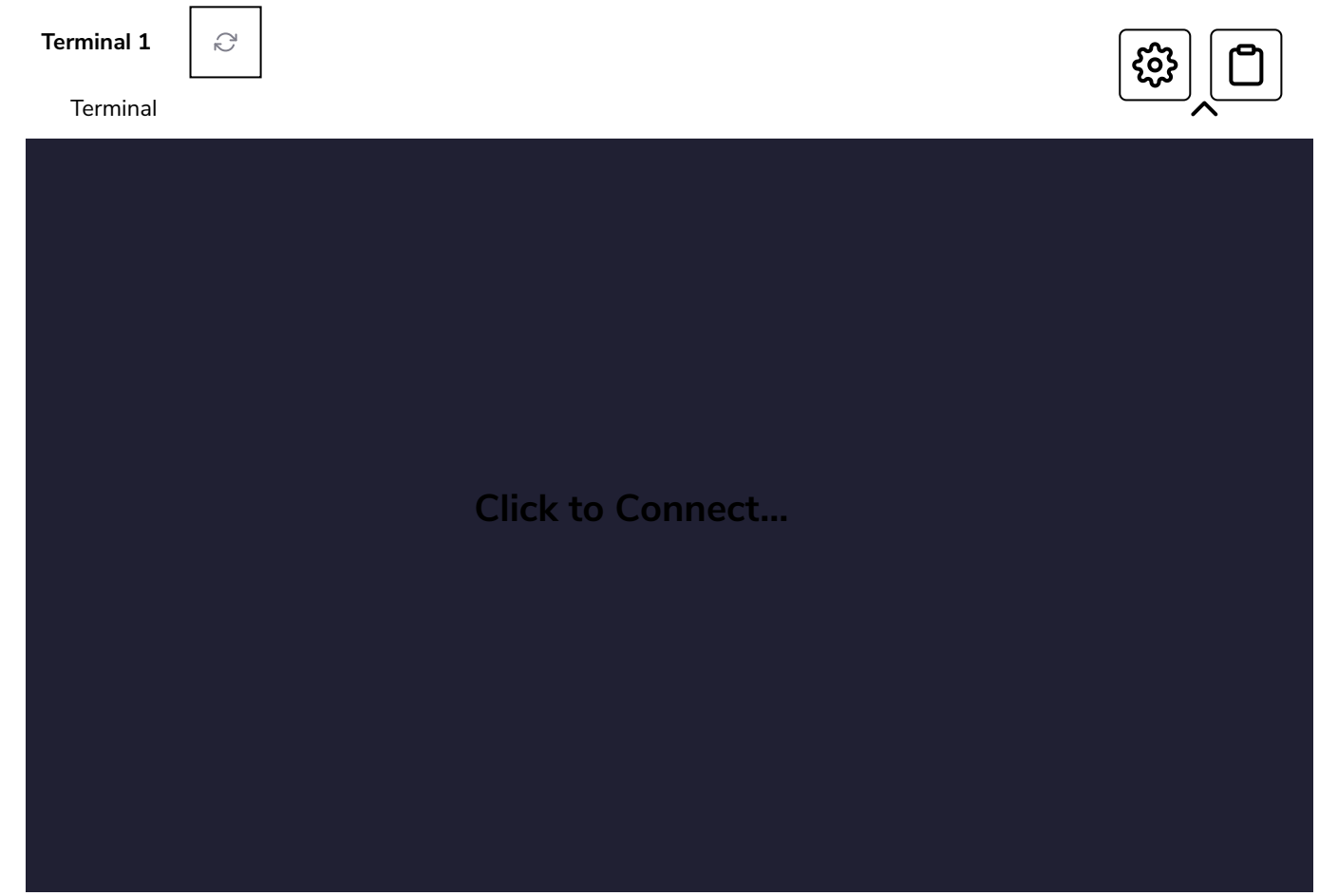
Learn about "git rebase" with the help of an example.

We'll cover the following

Rebase scenario

Let’s walk through the scenario from the previous lesson (https://www.educative.io/collection/page/10370001/6075350136651776/5237173092089856) with Git commands.

```
mkdir lgthw_rebase
cd lgthw_rebase
git init
echo A > file1
git add file1
git commit -am A
echo B >> file1
git commit -am B
echo C >> file1
git commit -am C
git checkout -b feature1
```



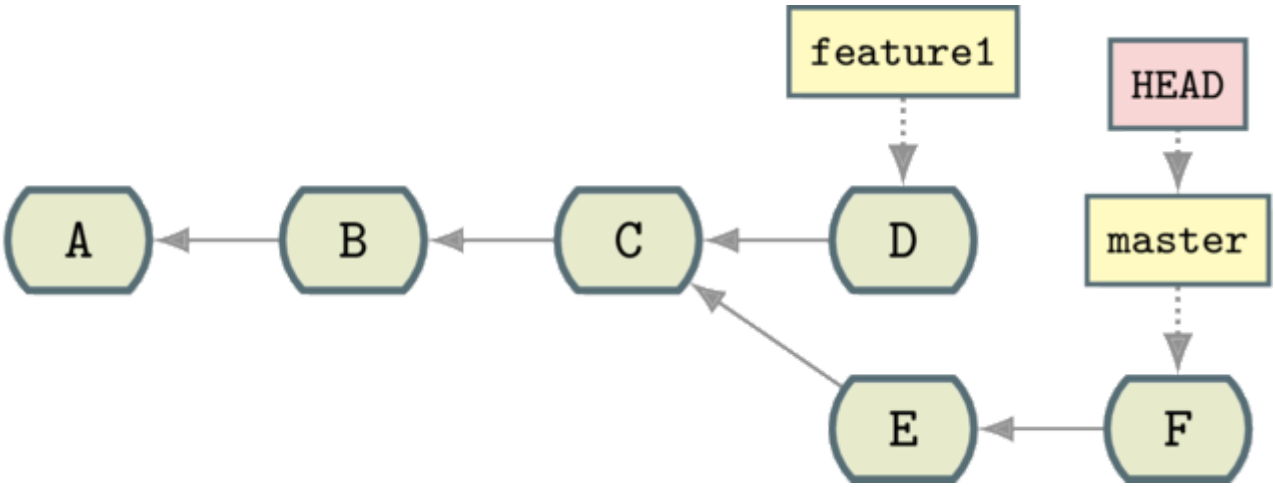
The `-b` flag

This is a shortcut that creates the branch and checks it out all at once. Using this means you don't get into the situation where you can create a branch but forget to explicitly check it out before committing more changes.

```
12  echo D >> file1
13  git commit -am D
14  git checkout master
15  echo E >> file1
16  git commit -am E
17  echo F >> file1
18  git commit -am F
19  git log --all --decorate --graph
```

State: Before rebase

You are now in this state:



Git rebase: before

Now go to the `feature1` branch and rebase:

```
20  git checkout feature1
21  git rebase master
```

Terminal 1

Terminal

Read the output carefully and follow the instructions. It's hard to follow, so don't be concerned if you don't get it the first time you try. Keep at it!

Stuck?
If you are struggling at this point, then you need to review this (https://www.educative.io/collection/page/10370001/6075350136651776/6272927020875776) chapter on merging and conflicts.

Resolve conflicts

The next command is there to indicate that you should resolve the conflicts mentioned.

```
22  vim file1
```

Terminal 1

Terminal

> **Note:** Since changes you made after `vim` are unique to your own, you can either continue using the above terminal for all the following commands in this lesson or use the following terminals with our `vim` edit.

`--continue` rebase session

Now that you've resolved the conflicts, you can continue the rebase session:

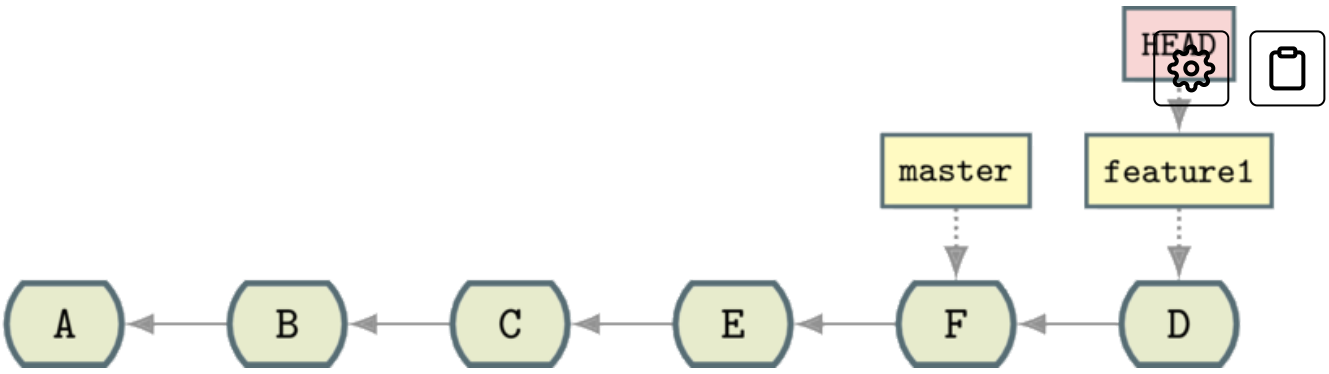
```
23  git add file1
24  git rebase --continue
    Applying: D
25  git log --all --decorate --graph
```

Terminal 1

Terminal

State: After rebase

The state now looks like this:



Git rebase: after

Merging and Fast-Forwarding

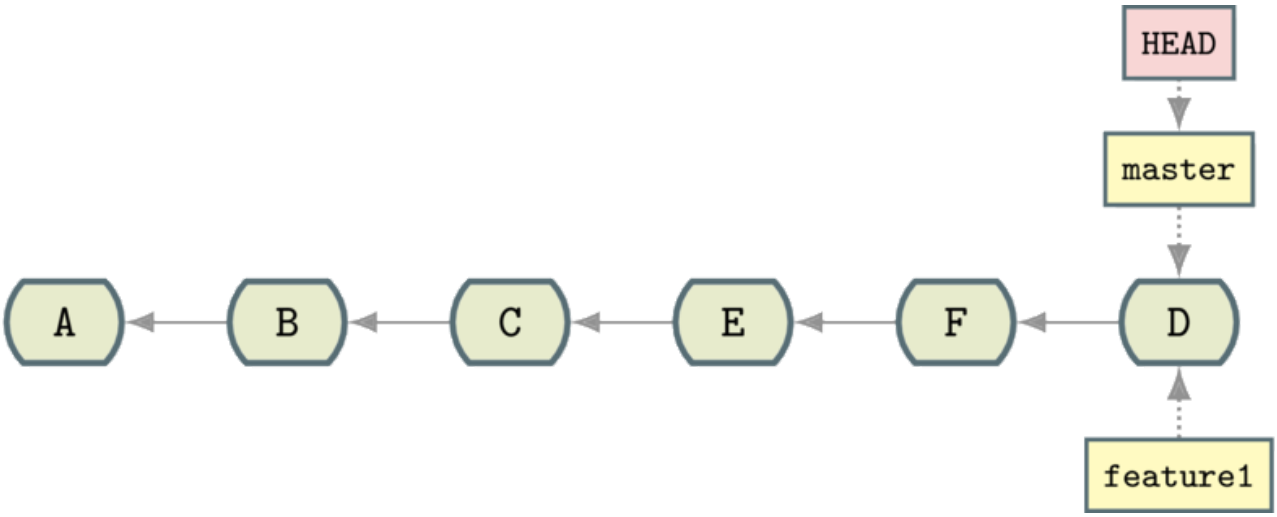
Now the changes are in one line you can merge the `feature1` branch into the `master` branch to move the `feature1` branch pointer along. Here we return to the `git merge` command you looked at earlier:

```
26  git checkout master
27  git merge feature1
28  git log --all --decorate --graph
```

Terminal 1

Terminal

And you end up in your desired state:



Final rebased state

Note that merging and rebasing are separate topics, but you return to it here because it's easiest to understand fast-forwarding in this context.

Fast-forwarding

What's interesting about the above is the output of the following command:

```
git merge feature1
```

```
root@educative:/lgthw_rebase# git merge feature1
Updating 21ff3b9..ea720d8
Fast-forward
 file1 | 4 ++++
 1 file changed, 4 insertions(+)
```

Fast-forwarding

Because the changes are in a line, there aren't any new changes that need to be made. The `master` branch pointer merely needs to be "fast-forwarded" to the same point as `feature1`!

Naturally, the `HEAD` pointer moves with the branch you're on (`master`).

Back

Outline of a Simple Rebase

Next

Introduction: Git Bisect

Mark as Completed

Report an Issue

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