Creating a Branch

Learn how to create a branch in Git.

We'll cover the following

- Creating a Git repository
- The git branch command
- Branch diagram

Creating a Git repository

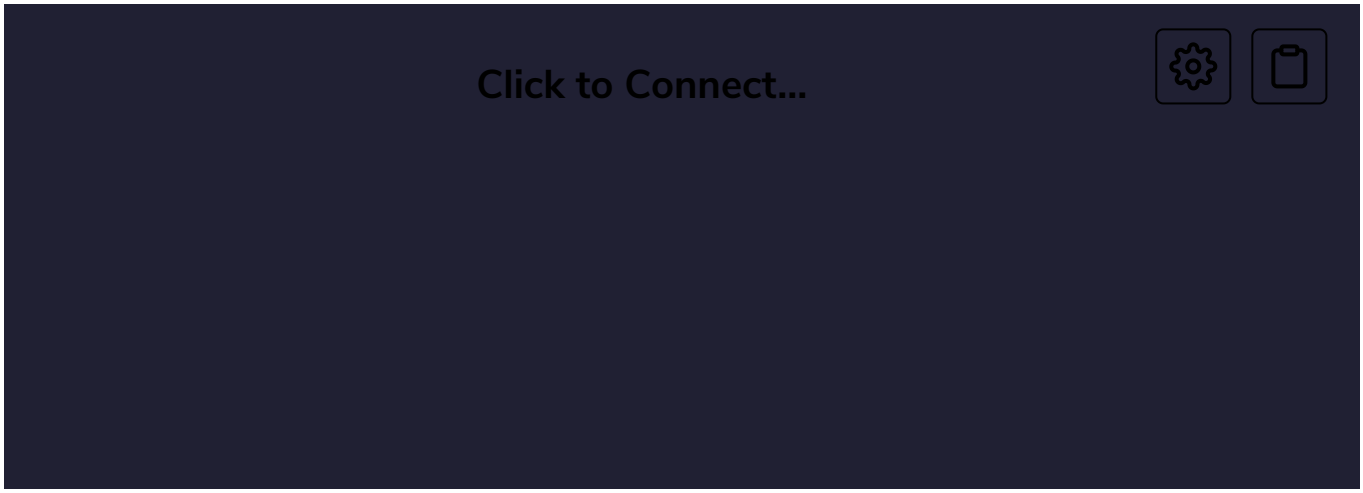
You will create a Git repository with a single file. This file will have separate changes made on two branches: `master` and `newfeature`.

Type the following command to create a simple Git repository:

```
mkdir lgthw_git_branch_1
cd lgthw_git_branch_1
git init
echo newfile > file1
git add file1
git commit -m 'new file1'
git status
```

Terminal 1

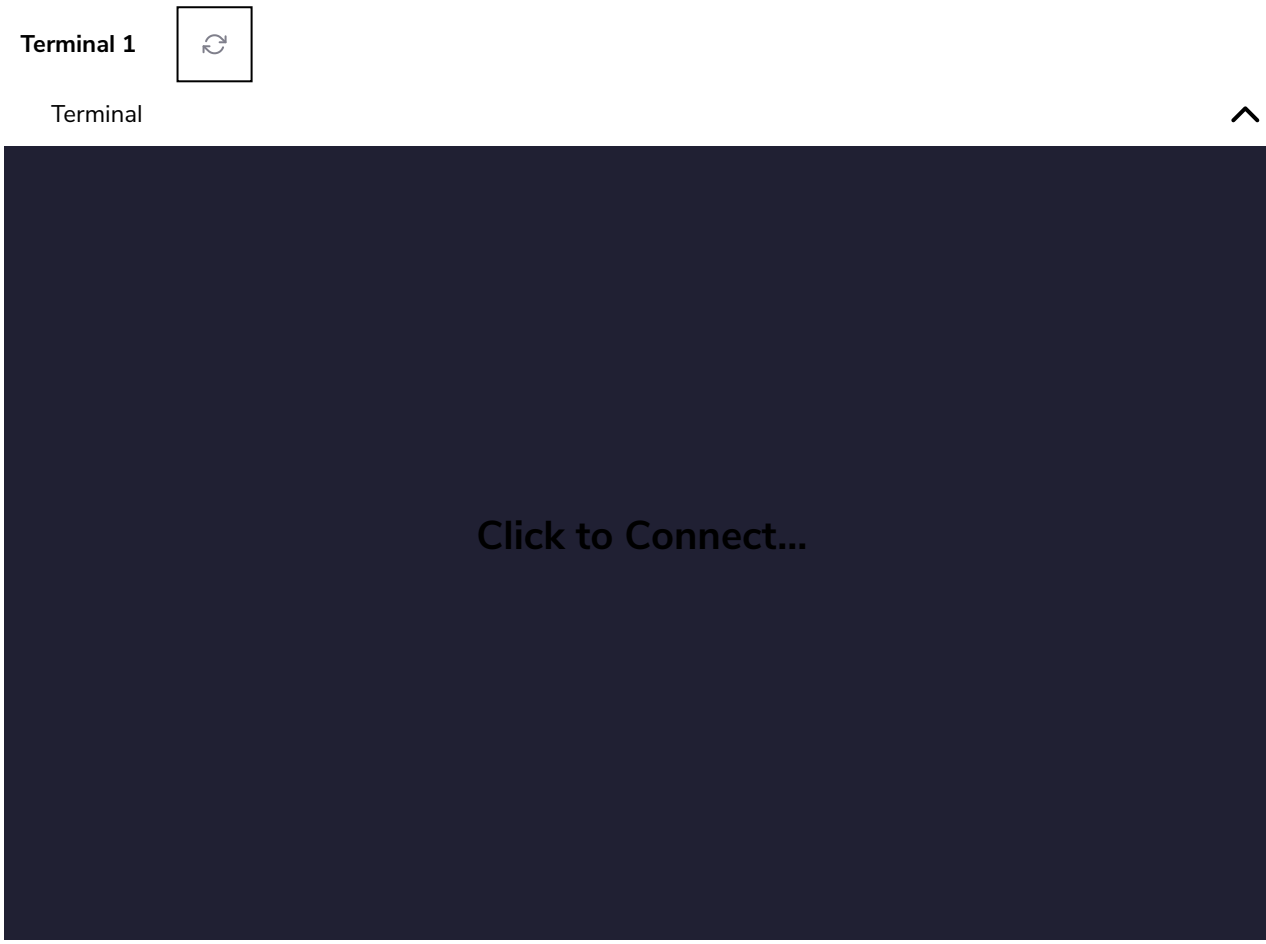
Terminal



The `git branch` command

To create a new branch, type the following commands:

```
8   git branch newfeature
9   git status
10  git branch
```



Now you have created a branch called `newfeature`.

What branch are you on now? Was that what you expected?

Add a commit on the `master` branch:

```
11  echo Line_master1 >> file1
12  git commit -am 'master change'
```

Terminal 1

Terminal

Click to Connect...

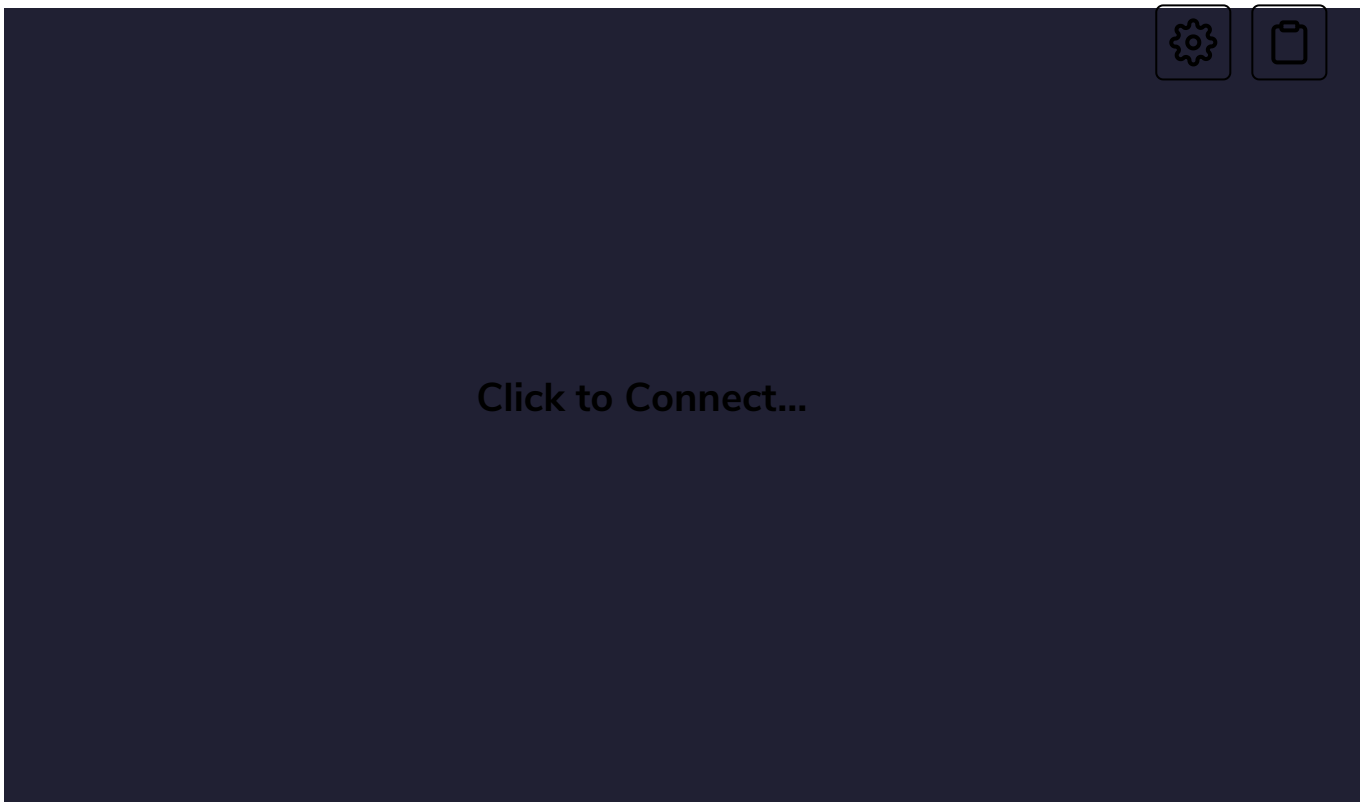
Type the following command. See if you can work out the difference between them:

```
13  git log --decorate --graph --oneline
```

The attentive learners will have noticed that I slipped in a new flag. The `--decorate` flag gives you useful information about the reference names (branches and tags) at various commit points. This is so handy. Adding it to `git log` is part of my muscle memory now.

Terminal 1

Terminal



Is `--decorate` the default?

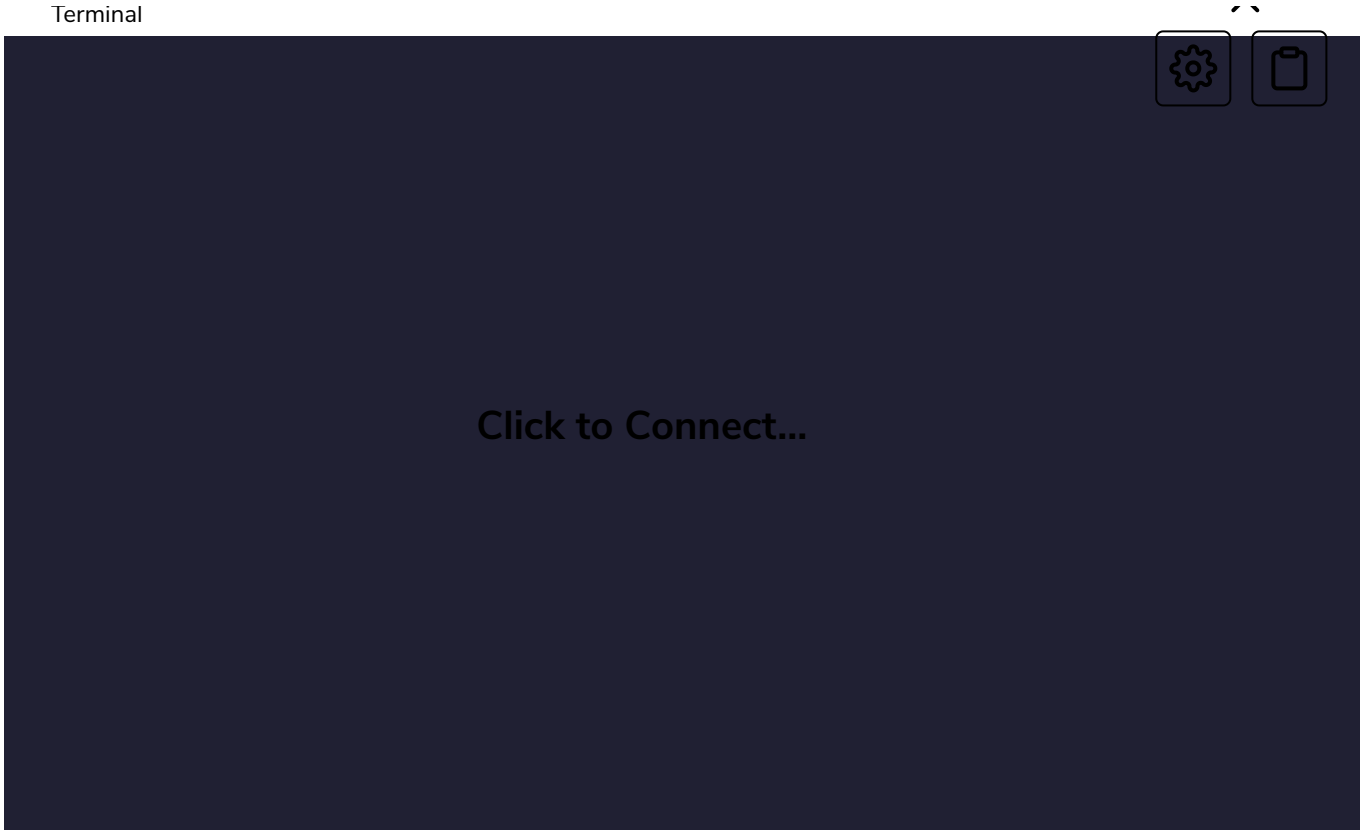
In recent versions of Git, it is the default to show `--decorate` if Git is writing to a terminal even if it's not supplied. If you output the `git log` command to a screen and redirect it to a file with the `>` operator, you might see a difference in content.

Type this in and think about what's going on:

```
14  git checkout newfeature
15  cat file1
16  echo Line_feature1 >> file1
17  git commit -am 'feature change'
18  git log --decorate --graph --oneline --all
```

Another flag (`--all`) was added there. As you've probably worked out, `--all` shows all branches, not just the branch you happen to be on.

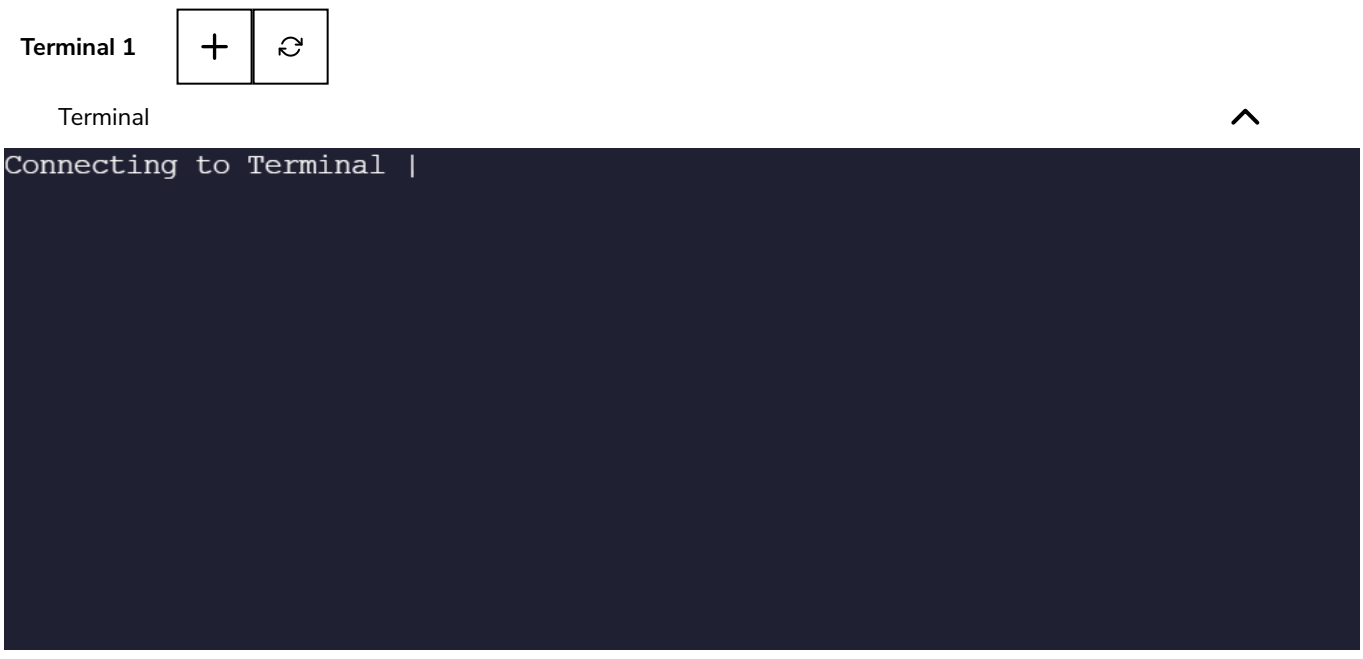
Terminal 1



This is also in my muscle memory, but sometimes it produces insane graphs I can't read, so I have to remember to remove it now and again.

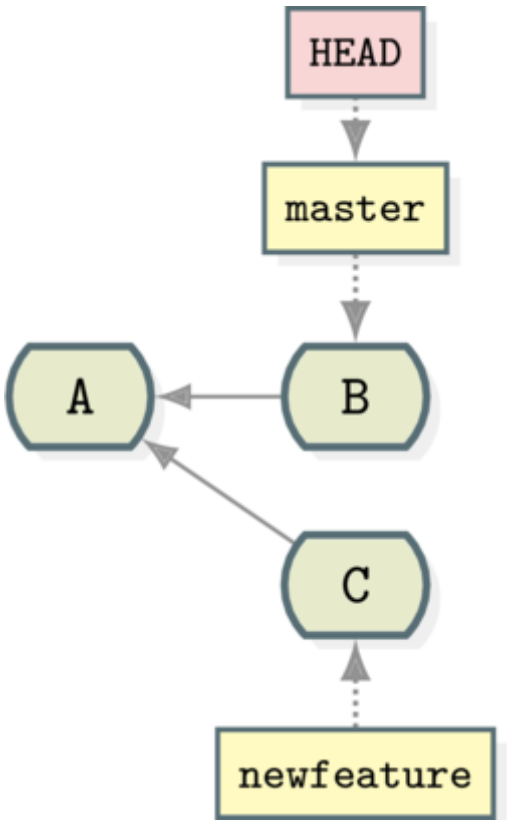
Back to the code. You have added a feature (i.e., a code change) to the `newfeature` branch. Go back to the `master` branch and check that the `file1` doesn't have the changes you made on the `newfeature` branch:

```
19  git checkout master
20  cat file1
```



Branch diagram

This diagram shows the final state of the commit tree:



A simple branched repository

This reflects the output of the last `git log` command. But note that it has a different orientation to the `git log` command. In the diagram, the commits are shown left-right in time order, while `git log` shows them from top to bottom.

Take a moment to be sure you've understood that difference in representation.

Differently-oriented and confusing Git diagrams are the norm in the Git world, and you will come across this challenge again later.

Note that the `HEAD` (and branch) moves forward with each commit.

The `HEAD` is where Git is pointed at right now in your repository, and the branch is where that branch reference is pointed to.

Back

Introduction: Git Branching

Next

Detached Heads

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/creating-a-branch__git-branching__learn-git-the-hard-way)