(/learn)

The 'git log' Command

Learn about the “git log”command and all flags associated with it.

We'll cover the following

`git log`

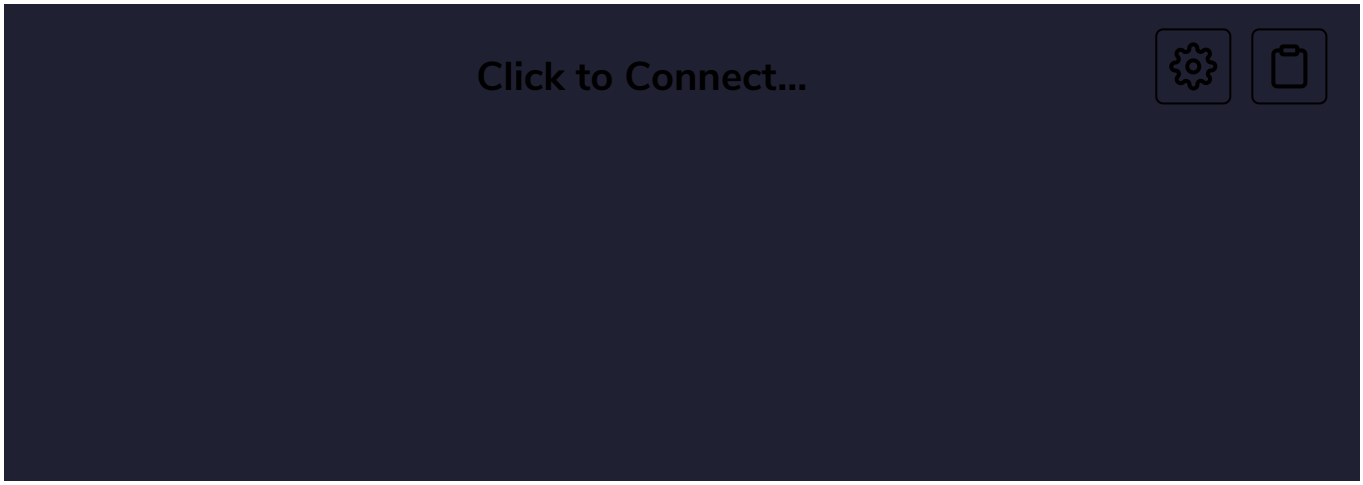
We have already seen `git log` earlier in the course.

The output is the most recent commit, down to the oldest from the current branch.

```
1   git log
```

Terminal 1

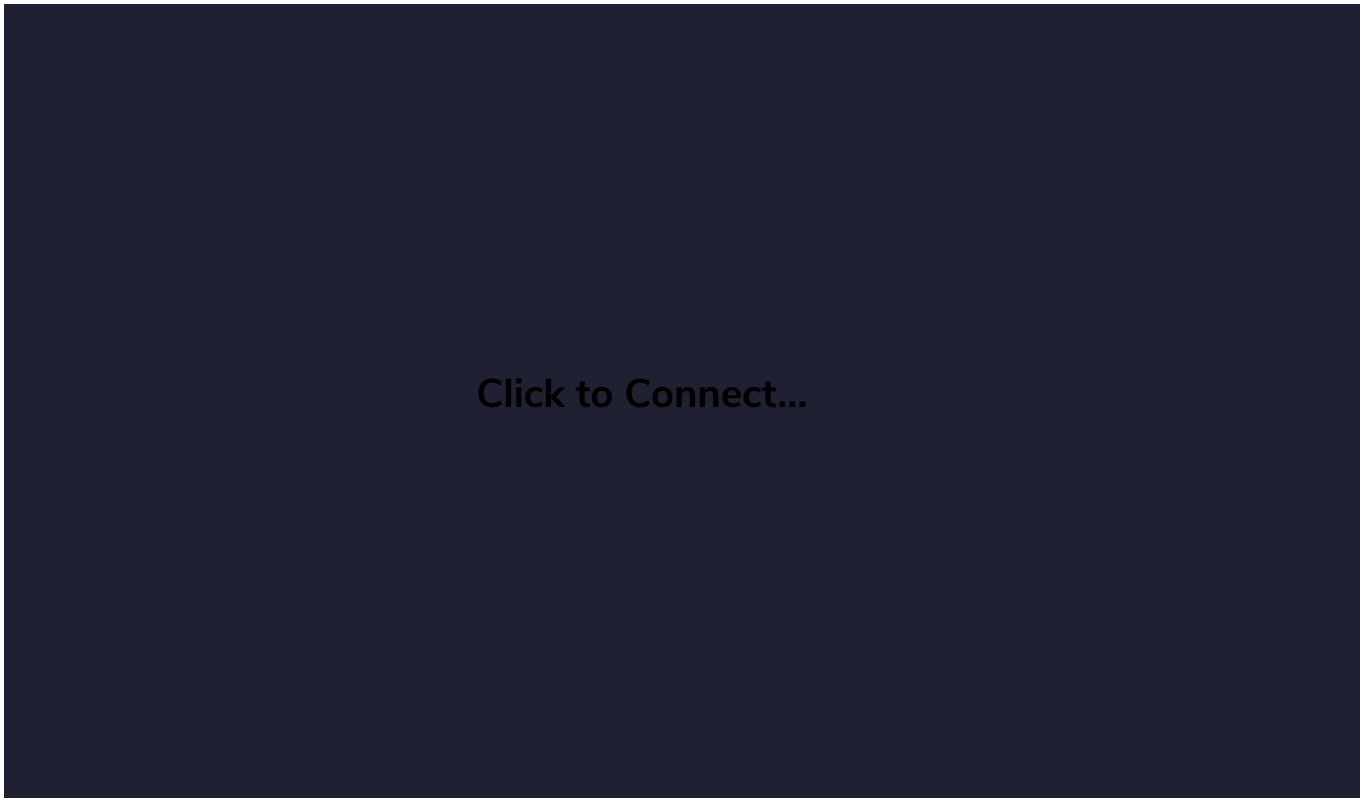
Terminal



`git log --oneline`

Most of the time, I don't care about the author or the date. In order to see more per screen, I use `--oneline` to only show the commit ID and comment per-commit.

```
2   git log --oneline
```



`git log --graph`

git log --graph

The problem with the above is that you only see a linear series of commits from the `HEAD` and do not get a sense of what was merged in where.

```
3   git log --oneline --graph
```

Terminal 1

Terminal

You can see where merges take place and what commits were merged. If you recall the chapter on rebasing (https://www.educative.io/collection/page/10370001/6075350136651776/6228325110906880), then you know it is preferable to rebase your branches to the main branch and fast-forward on a merge; you get a cleaner and simpler history.

git log --all

By default, you only get the history leading up to the `HEAD` (i.e. where you are currently in the Git history). Often I want to see all the branches in the history, so I add the `--all` flag.

```
4   git log --graph --oneline --all
```

Note that for this repository, the output might look very similar to the previous one. If you want to prove there's a difference, then try this:

```
5   git log --graph --oneline --all > all_output
6   git log --graph --oneline > noall_output
7   diff all_output noall_output
```

Terminal 1

Terminal

git log --decorate

That's great, but I can't see what branch is where! This is where you use the `--decorate` flag.

```
8   git log --graph --oneline --all --decorate
```

Terminal 1

Terminal

Now you're cooking with gas! Each remote or type of branch/tag is shown in a different color (even stashes!). On my terminal, remotes are in red, HEAD is blue, local branches are in green, and stashes are in pink.

If you want, you can show the ref name on each line by adding `--source`. But I usually find this to be an overkill:

```
9   git log --graph --oneline --all --decorate --source
```

git log --simplify-by-decoration

If you're looking at the whole history of your project, you may want to only see the significant points of change (i.e., the lines affected by `--decorate` above) to eliminate all the intermediary commits. This is perfect for getting an overview of the project's shape as a whole.

```
10  git log --graph --oneline --all --decorate --simplify-by-decoratio
n
```

Terminal 1

Terminal

git log --pretty

git log --pretty=*

When viewing the whole history of the project in this way, you might want to re-introduce the date information with the `--pretty=*` flag:

```
11  git log --graph --oneline --all --decorate --simplify-by-decoratio
n --pretty='%ar %s %h'
```

Terminal 1

Terminal

This gives a formatted output, showing (in this case) the relative timestamp (`%ar`), the commit subject (`%s`), and the short hash (`%h`).

You can even see the abstract shape of the Git repository by not printing any details!

```
12  git log --graph --oneline --all --decorate --simplify-by-decoratio
n --pretty=
```

The output of this command depends on the Git version, so it might look different to you.

Back

A Realistic Log History

Next

Introduction: Squashing Commits

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/the-git-log-command__git-log__learn-git-the-hard-way)