What Are Some Important Git Commands?

Done

Checking If Git Is Installed

If you're following along on a local set up (you don't have to, but just in case you are), start by checking if Git exists on your system with the following command:

```
git --version
```

Setting Up And Starting A New Git Repository

To mark a directory as a Git working directory, call the following command in that directory:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
git init
```

Git should be tracking any changes we make within this folder now. So let's add a file to see if Git notices anything with:

```
git status
```

```
git init
touch index.html
git status
```

So Git noticed all of the files in the directory!

Also, note that each of our coding playgrounds is like an individual virtual machine, that gets created and destroyed upon execution which is why we have to initialize a repo each time! This is also why so many 'random' files already exist. Also, to make the console output less cluttered, pass the quiet flag like `-q`.

What Is The Staging Area?

Staging is just a step that has to be done before the final commit. Imagine it to be a box that your changes are put into before they are committed.

Adding New Files To The Staging Area

Add new files to the staging area with,

```
git add folder/that/contains/files
```

and a git status call on line 4 would show us all the changes to be committed.

```
git init -q
touch index.html
git add .
git status
```

Committing The Files

When we commit files, we submit it to be added to the main branch of code. Commit the files with:

```
git commmit -m "a message to commit with"
where the string after -m is a message that describes the commit. An example would be
```

```
git commmit -m "fixing the bug number 209"
```

```
git init -qtouch index.htmlgit add .git commit -m "Our first commit!"
```

Checking Commit History

To print out git's commit history, type:

```
git log
```

Output example:

```
__ed_create_user.sql
__ed_destroy_user.sql
__ed_javaRunner.sh
__ed_script.sh
__ed_sql_runner.sh
index.html main.sh output

commit 3094b0bcf3483de4955be7c4c15a0b65c3e765b3
Author: Your Name <you@example.com>
Date: Wed Jan 2 07:38:11 2019 +0000 Our first commit!
```

Where the first few lines represent files that were modified or added and the numbers after the `commit` field represent the hash value of the commit (a unique string that identifies the commit). The `Author` and `Date` fields contain information about the author, the time of commit, and the message the author sent with the commit.

```
git init -q
touch index.html
git add .
git commit -m "Our first commit!" -q
git log
```

Reverting To A Previous Commit

To revert your working directory to any previous commit, type the command:

```
git checkout hashvalue
```

We can't try reverting with hash values on our platform because a different one is generated for each time the code is run. So we'll specify the syntax to revert to the previous state. Notice how even though `index.html` is removed on line 5, it is restored with git checkout.

```
git init -q
touch index.html
git add .
git commit -m "Our first commit!" -q
rm index.html
git checkout -- .
ls
```

Back to Top

Feedback

Tell us what you thought about the chapter: What are some important git commands?