Command Line and Git

GENERAL ASSEMBLY

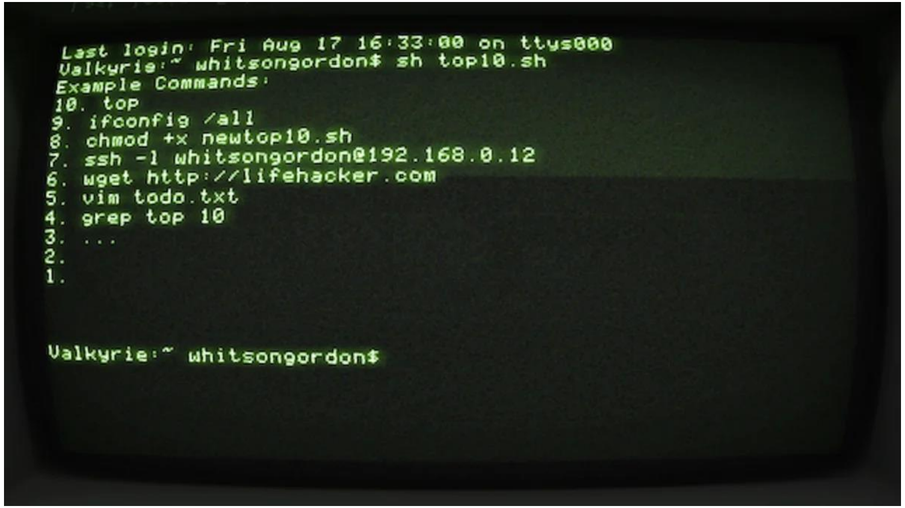
Learning Objectives

- Part 1: The UNIX Command Line
  - What is it and why use it?
  - Movin' around
  - Lookin' at things
  - Makin' things
- Part 2: Git
  - What is version control, and why use it?
  - The Git workflow
  - How ***we*** use Git in our class

CLI vs GUI

It used to be computers didn’t have a **graphical user interface (GUI)**. You had to do everything via the **command line interface (CLI)**.

Today we take GUIs for granted. Most computer users, if they even know the command line exists, are confused and even a little scared by it. But not you!



```
Last login: Fri Aug 17 16:33:00 on ttys000
Valkyrie:~ whitsongordon$ sh top10.sh
Example Commands:
10. top
9. ifconfig /all
8. chmod +x newtop10.sh
7. ssh -l whitsongordon@192.168.0.12
6. wget http://lifehacker.com
5. vim todo.txt
4. grep top 10
3. ...
2.
1.


Valkyrie:~ whitsongordon$
```

https://i.kinja-img.com/gawker-media/image/upload/c_fill,f_auto,fl_progressive,g_center,h_675,pg_1,q_80,w_1200/17waftgjfrx4pjpg.jpg

What is the Command Line?

We’ll use the CL for all sorts of things.

Why do you think it might be more beneficial to use a text-based CLI rather than a GUI?

https://securecdn.pymnts.com/wp-content/uploads/2019/12/hacker-Apple-Turkish-NCA-investigation.jpg

Terminal-ogy

- The **shell** is the specific command line language you're typing in
  - The go-to industry standard is **bash**, so that's what we'll use
  - … unless you're using the newest version of MacOS, in which case you'll be using **Zshell**, which is nearly identical

- The **terminal** is the program you use to emulate the shell
  - If you're on MacOS, that program is simply called **Terminal**, but another popular third party choice you can download is **iTerm2**
  - If you're on Windows, that program is **Git Bash**, although some experienced users might prefer the **Windows Subsystem for Linux (WSL)**

Let’s get started!

Open up those terminals!

So you’re dropped into a terminal...

Where are we?

pwd = print working directory (where am I right now?)

(We’ll all see something different)

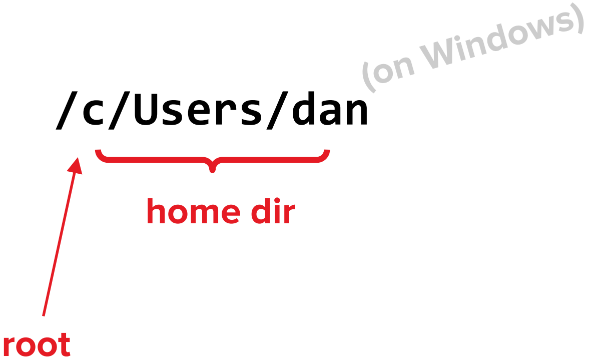
File Paths: root

In UNIX-based systems (ie, Mac and Linux), your folder's file tree begins at **root**. The **root directory** is the folder on your computer that contains *everything*. It's denoted simply by a **slash (/)**.

In Windows, Git Bash will emulate this, but it's not *really* true. Your C:\ drive is mounted in **/c**.

File Paths: home

Each user on a computer has a **home directory**, which is where you start out when you open your terminal. It is often denoted simply with a **tilde (~)**. When I typed, pwd, I saw:



```
/c/Users/dan
```

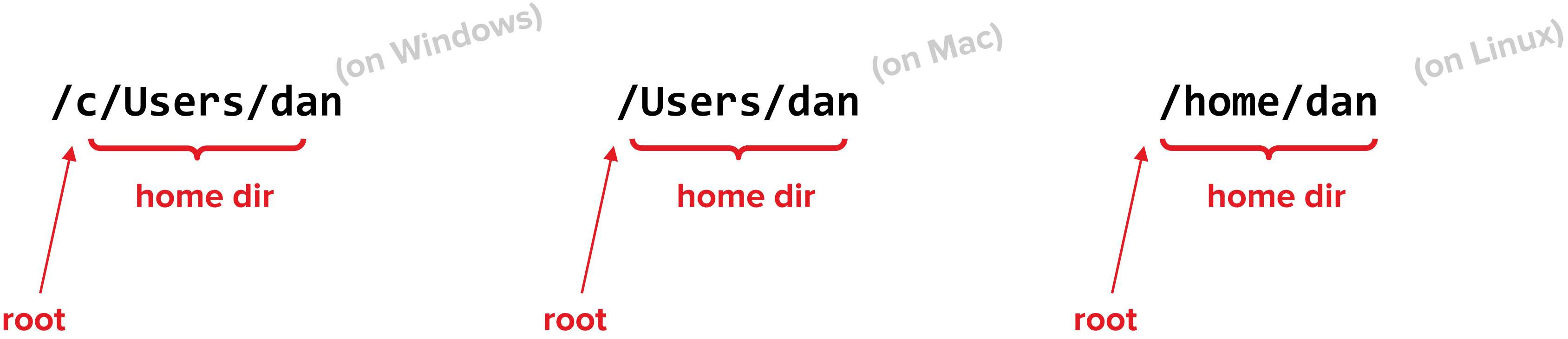
(on Windows)

home dir

root

File Paths: home

Each user on a computer has a **home directory**, which is where you start out when you open your terminal. It is often denoted simply with a **tilde (~)**. When I typed, pwd, I saw:



(on Windows)

`/c/Users/dan`

home dir

root

(on Mac)

`/Users/dan`

home dir

root

(on Linux)

`/home/dan`

home dir

root

Let’s Walk and Talk

How do we move around our computer?

cd = change directory = move to a certain folder

Absolute Paths

You can **cd** into a specific directory if you know its *exact* location. The exact location, starting with root, is a folder's **absolute path**. For example:

```
cd /Users/dan/Documents/project
```

Absolute path

or...

```
cd ~/Documents/project
```

**Absolute path
(bash expands ~
to /Users/dan)**

Relative Paths

Sometimes you don't know the *exact* location of a folder, but you do know it *relative* to where you currently are. If this is the case, you can also use a **relative path**. For example, if we're currently in our home directory,

instead of

```
cd ~/Desktop/project
```

Absolute path

we can write

```
cd Desktop/project
```

Relative path
(to the working
directory)

Relative Paths

And again, if we're on our Desktop, we can write

```
cd project
```

But how do I get back up to my desktop? Or my home directory? The notation to "go up" a folder is the **double dot (..)**. And so, from inside this project folder, to go back up to the desktop, you would type:

```
cd ..
```

(go up one level)

And to go up to your home directory:

```
cd ../..
```

(go up two levels)

Shortcuts

cd (with no path given)	Go to home
cd ~	Go to home
cd ..	Go up
cd .	Do nothing (**.** = “this folder”)
cd -	Go back to last directory

Look Around

The **ls** command is used to **list** our all the files and folders in a given directory.

You Try:

- Go to your Downloads folder.
- Look at what is inside it.
- Go back to your home folder.

Makin’ Stuff

Creating Directories

You can use the **mkdir** command to create a new directory

You Try:

Make a folder on your desktop called **tutorial** and then navigate into it.

Let’s Explore our Development Environment

Make sure you’re in this new directory. Let’s run the **jupyter notebook** command to execute the Jupyter Notebook software.

Let’s play around here and make a quick notebook.

Let’s make a quick empty file

You can use the **touch** command to make an empty file.

Use **touch** to make a **hello.py** and **goodbye.py** file in our tutorial folder.

Let’s Make a Python Script!

You can use the `echo` command to send text to stdout (“standard output”).

```
echo 'Hello, world!’
```

By default, stdout is the terminal – so it displays on your screen.

Let’s Make a Python Script!

We can redirect stdout elsewhere by using `>` … for example to a file! Let’s create our first Python script this way.

```
echo 'print("Hello, world!")' > hello.py
```

This is one of many reasons why Linux is so powerful. You can send the output of any program directly to a file -- or even to another program as its input!

Hello, World!

Not only is Python a programming language… it's also a program! Actually, it's a program that runs programs! We can run this file with the **`python`** command (**`python3`** on some Linux and MacOS versions).

To run our program, we can simply run

```
python hello.py
```

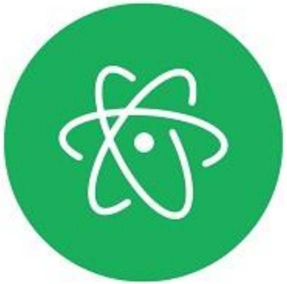
But what if our file changes? How can we keep track…?

Cheat Sheet

`cd /path/to/folder`	Change directory (move)
`pwd`	Print working directory (where am I now?)
`ls`	List everything where I am
`mkdir folder-name`	Make a folder called **folder-name**
`touch file-name.py`	Make an empty text file called **file-name.py**
`echo string`	Sends **string** to stdout.

Optional: Let’s Edit!

To edit a plain old text file, we typically would use a **text editor**. There are millions of them out there, and experienced programmers debate which is best very fiercely. Right now, the three most popular ones are **Atom**, **VS Code**, and **Sublime**. We asked you to install Atom, so that’s what we’ll work with today.



Open up Atom using your GUI if you’d like. An additional way to do is to run **`atom .`** from the command line.

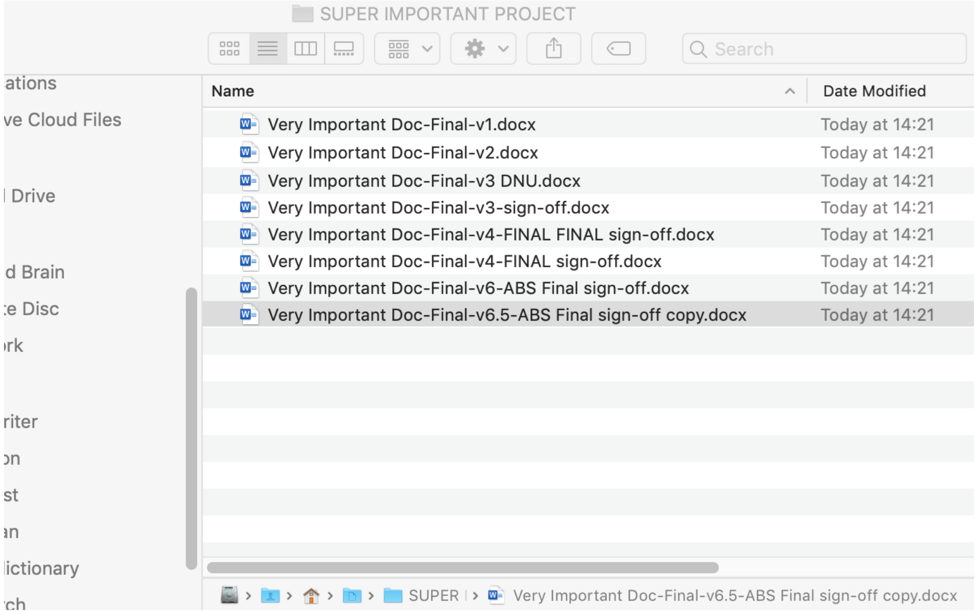
Version Control with Git

What is Git?

Git is **version control software**. Why would you need such a thing?

What is Git?

Git is **version control software**. Why would you need such a thing?



Name	Date Modified
Very Important Doc-Final-v1.docx	Today at 14:21
Very Important Doc-Final-v2.docx	Today at 14:21
Very Important Doc-Final-v3 DNU.docx	Today at 14:21
Very Important Doc-Final-v3-sign-off.docx	Today at 14:21
Very Important Doc-Final-v4-FINAL FINAL sign-off.docx	Today at 14:21
Very Important Doc-Final-v4-FINAL sign-off.docx	Today at 14:21
Very Important Doc-Final-v6-ABS Final sign-off.docx	Today at 14:21
Very Important Doc-Final-v6.5-ABS Final sign-off copy.docx	Today at 14:21

SUPER › Very Important Doc-Final-v6.5-ABS Final sign-off copy.docx

Screenshot from Charlie Rice

Git solves this (for text files)!

Git was created by legendary programmer **Linus Torvalds**, the same man who brought us Linux.

It was actually written in 2005 as a tool to help maintain the Linux kernel itself! He named it Git, since everyone was calling him a git (slang for a grumpy old man).

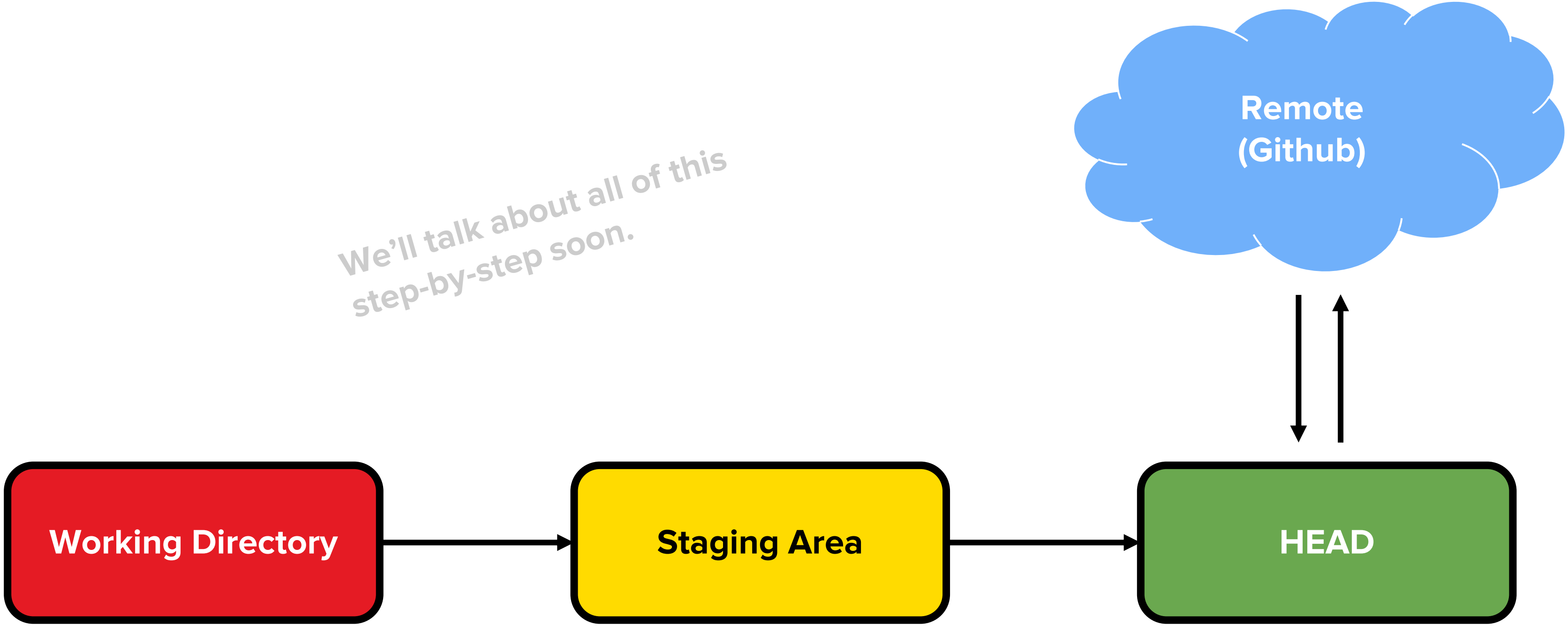
Linus + UNIX = Linux

https://cdn.britannica.com/99/124299-050-4B4D509F/Linus-Torvalds-2012.jpg

Too Many Gits

Git	A distributed version control system.
Git Bash	A very bad name for a Bash terminal emulator. A better name might be “(Minimal) Bash with Git.”
Github	A popular website for hosting Git repositories. Think “Instagram for programmers.”
Github Enterprise	A commercial version of Github. General Assembly pays a lot of money annually for our own private Github Enterprise server.

The Git Workflow



We'll talk about all of this step-by-step soon.

First: Let’s Wrangle a Repo

There are two ways to create a repository:

1. Start one from scratch via the command line.
2. Create one using Github and “clone” it onto our own machines.

First: Let’s Wrangle a Repo

There are two ways to create a repository:

1. Start one from scratch via the command line.

2. **Create one using Github and “clone” it onto our own machines.**

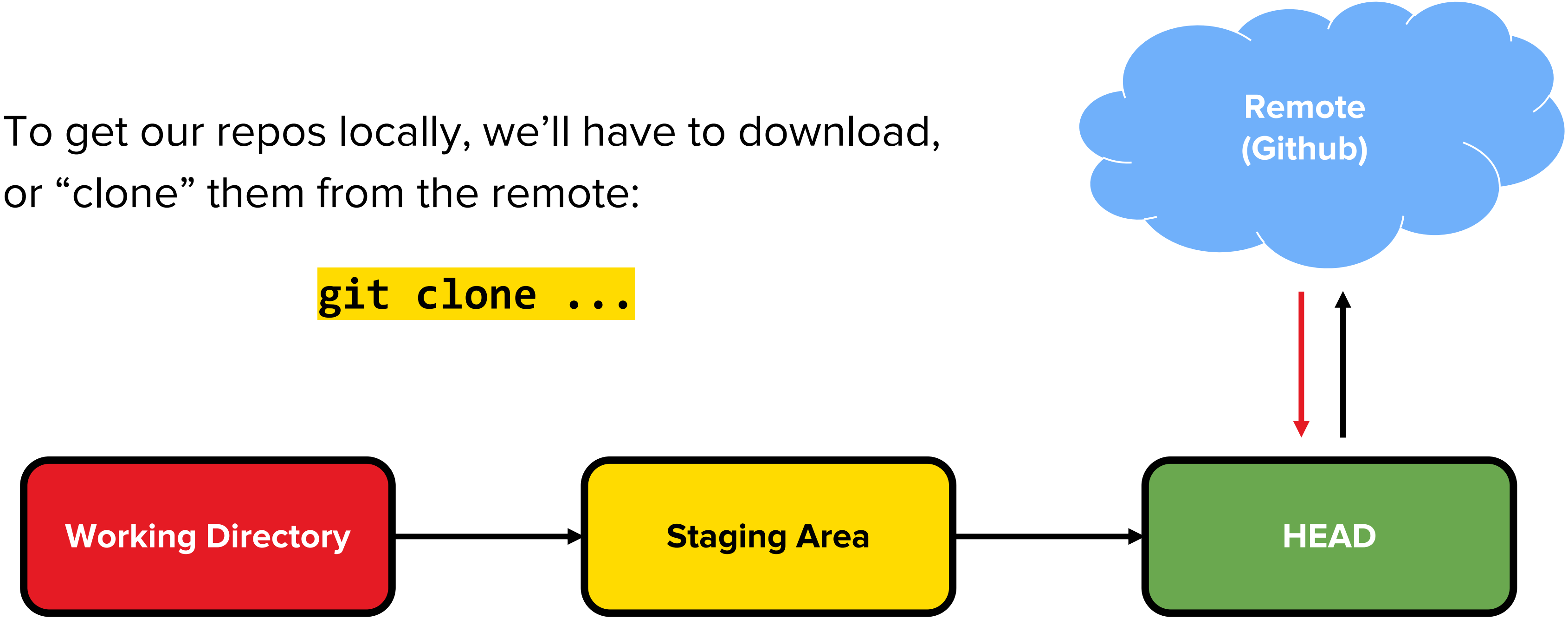
Let's do that!

Let's create a repo named **my-first-repo** now!

Bringing Things Down to Local

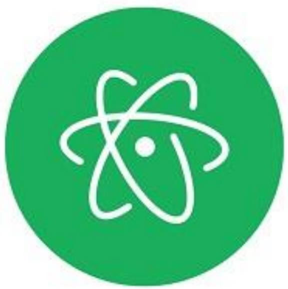
To get our repos locally, we'll have to download, or "clone" them from the remote:

```
git clone ...
```



Be the change you wish to see in the repo

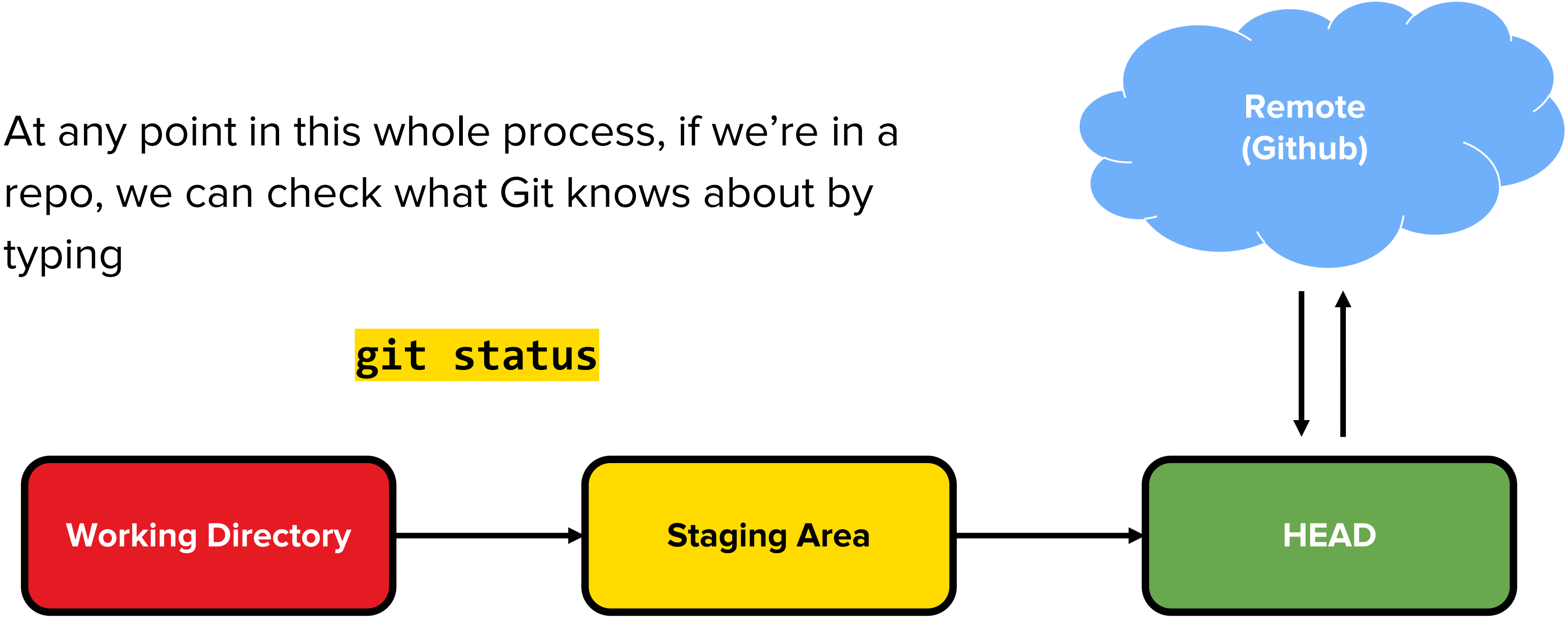
Let’s create a quick **greet.py** file in our repo.



Status Checking

At any point in this whole process, if we're in a repo, we can check what Git knows about by typing

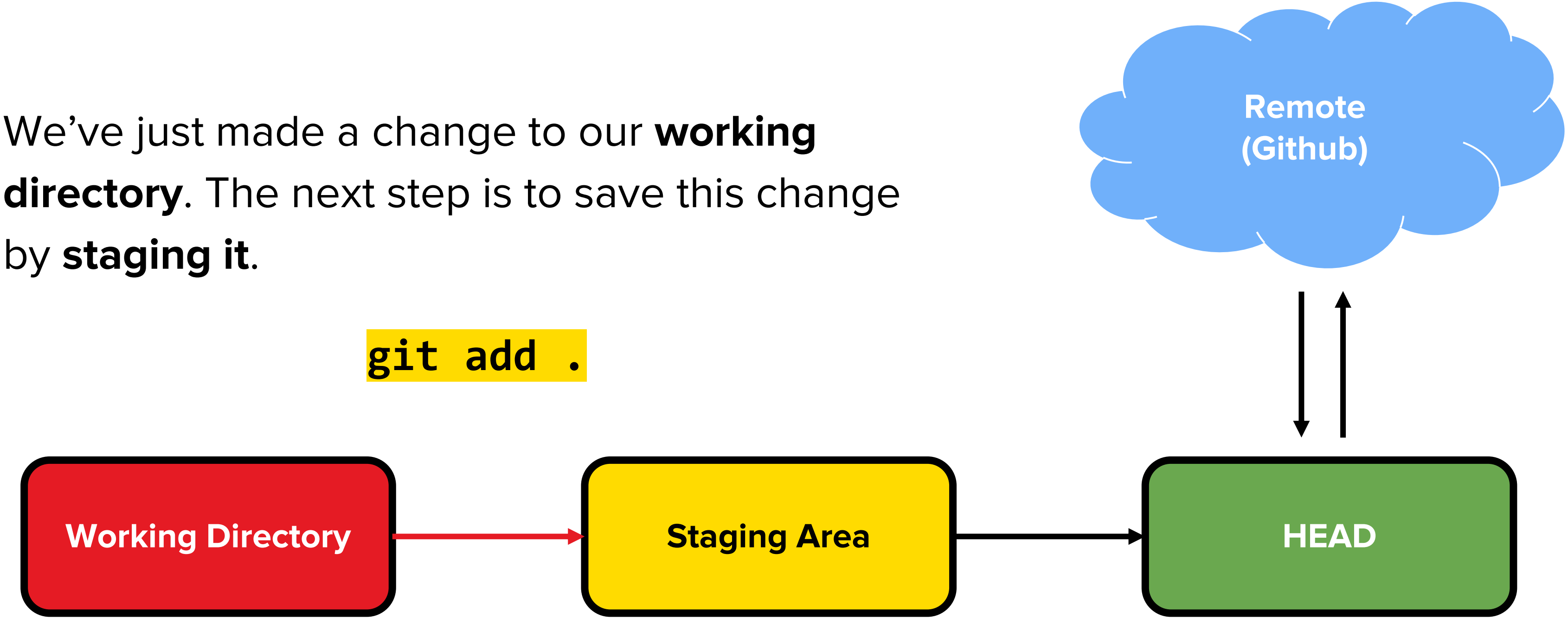
```
git status
```



Staging Changes

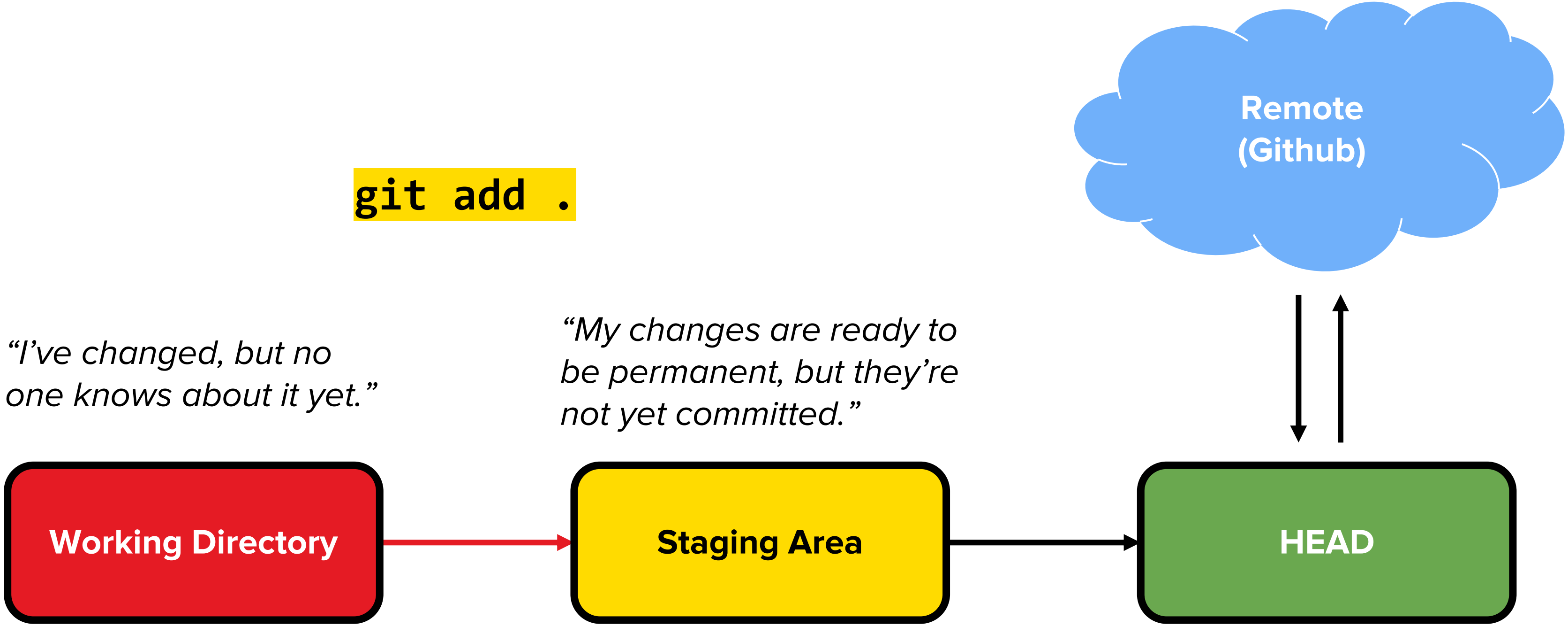
We’ve just made a change to our **working directory**. The next step is to save this change by **staging it**.

`git add .`



Staging Changes

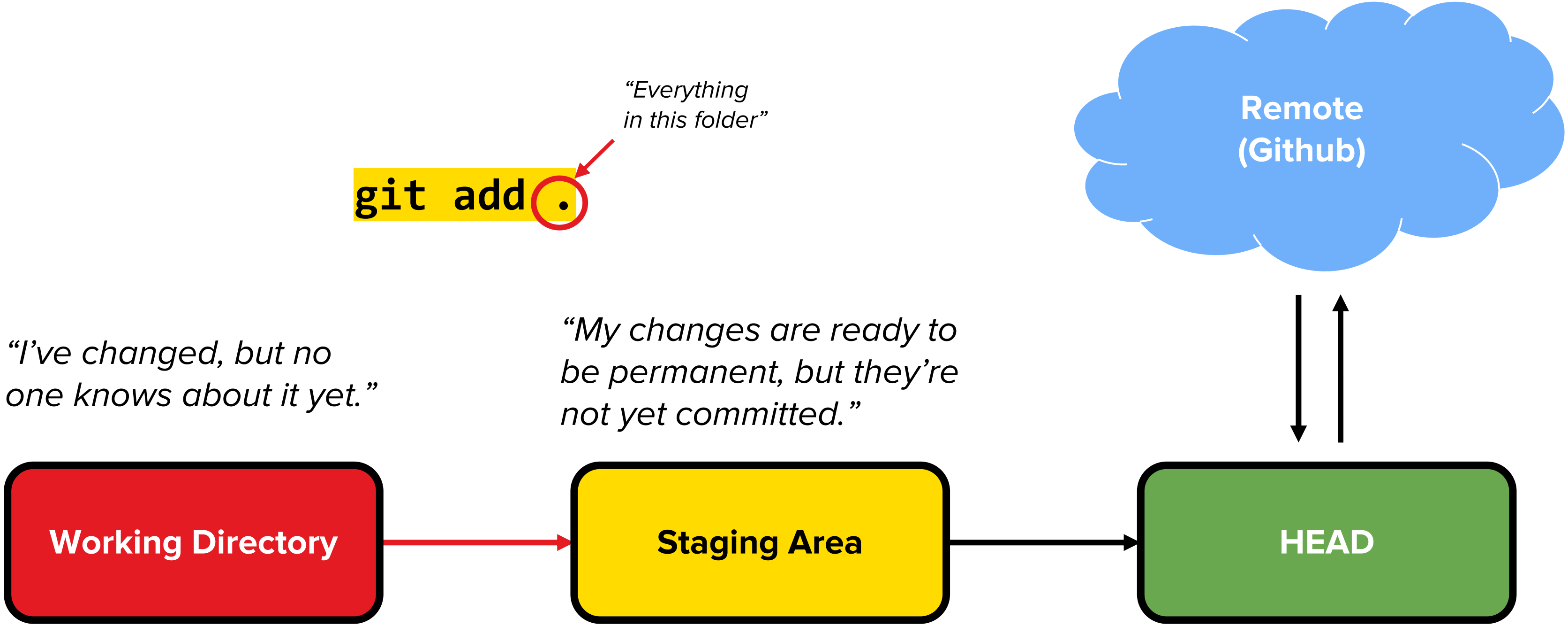
```
git add .
```



“I’ve changed, but no one knows about it yet.”

“My changes are ready to be permanent, but they’re not yet committed.”

Staging Changes



`git add .`

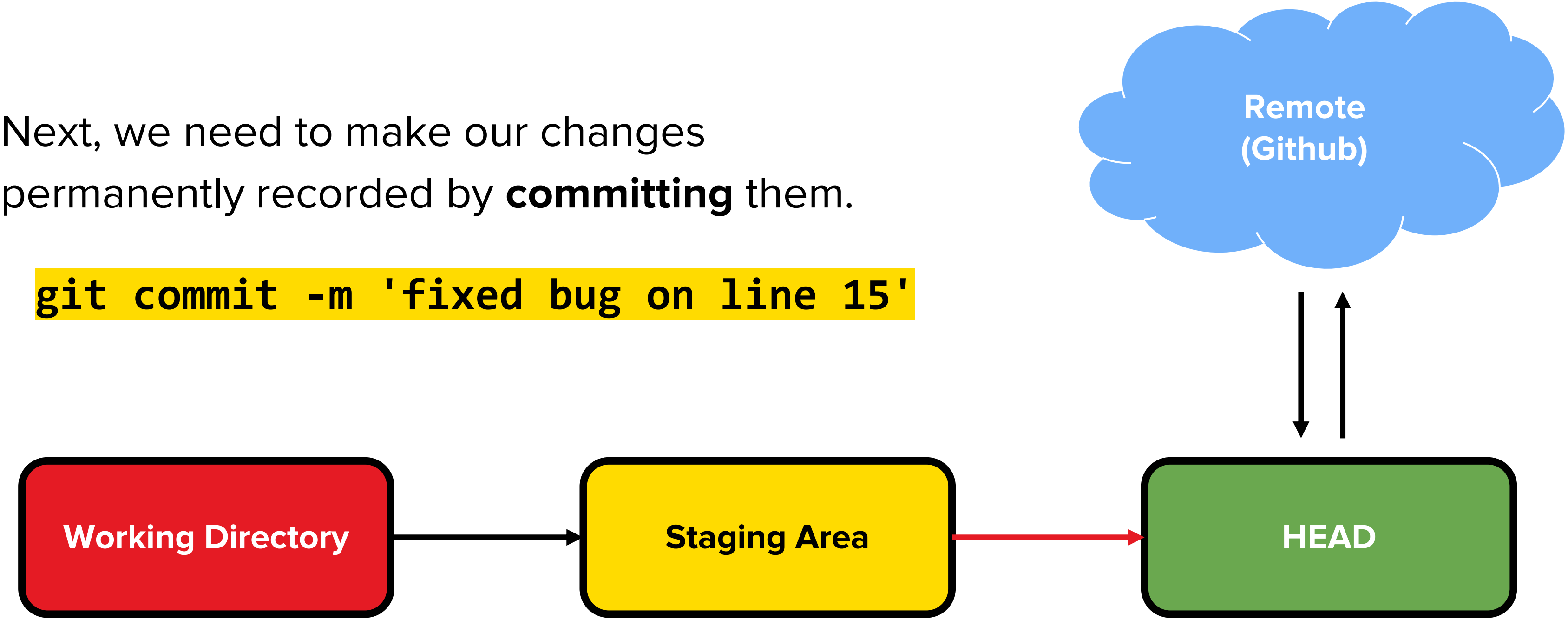
"I've changed, but no one knows about it yet."

"My changes are ready to be permanent, but they're not yet committed."

Committing to Change

Next, we need to make our changes permanently recorded by **committing** them.

```
git commit -m 'fixed bug on line 15'
```



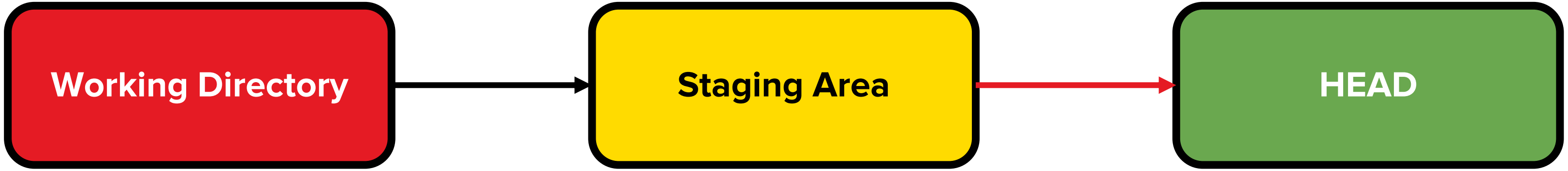
Committing to Change

```
git commit -m 'fixed bug on line 15'
```

“I’ve changed, but no one knows about it yet.”

“My changes are ready to be permanent, but they’re not yet committed.”

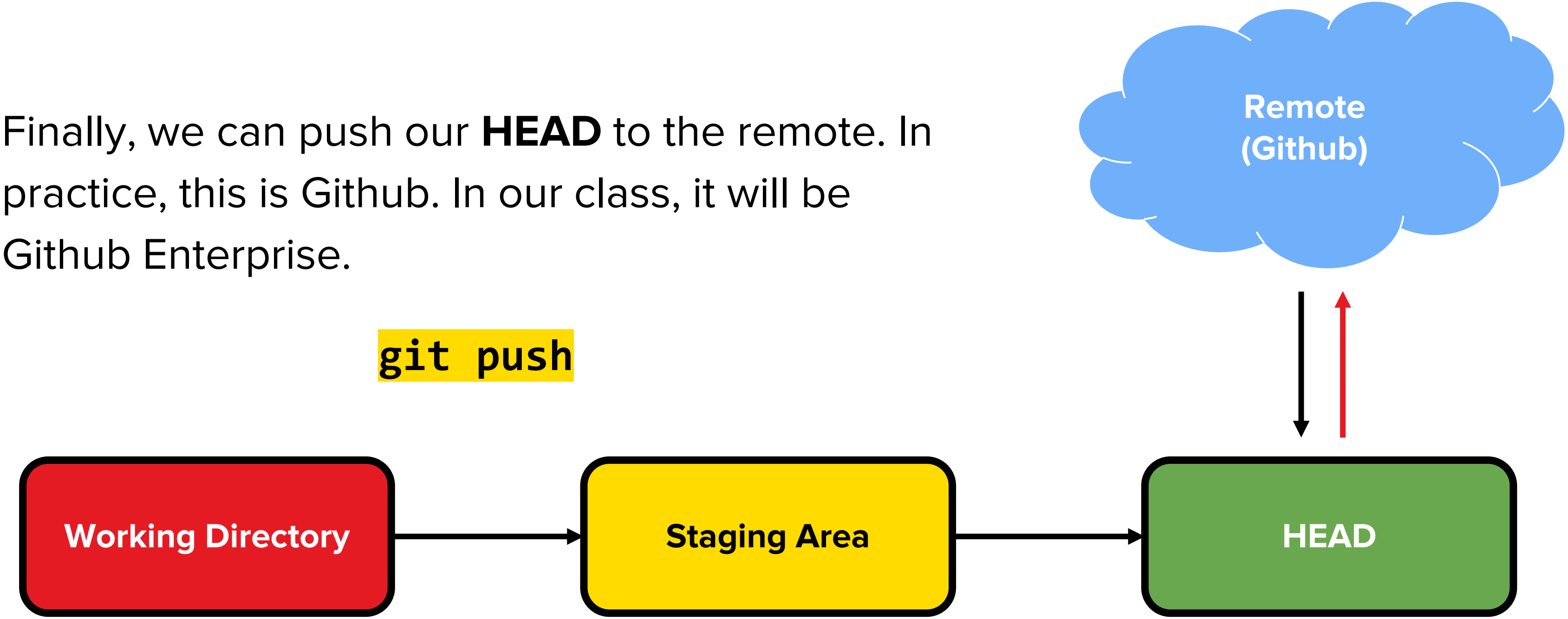
“My changes have been permanently committed, I’m ready for the world to see!”



Publishing our Work

Finally, we can push our **HEAD** to the remote. In practice, this is Github. In our class, it will be Github Enterprise.

```
git push
```



Git Workflow

`git add .`	Add changes to staging area
`git commit -m ‘msg’`	Commit changes permanently
`git push`	Push committed changes to Github

`git status`	Ask Git what’s going on. You can do this anytime, and should do it often!

Git Workflow

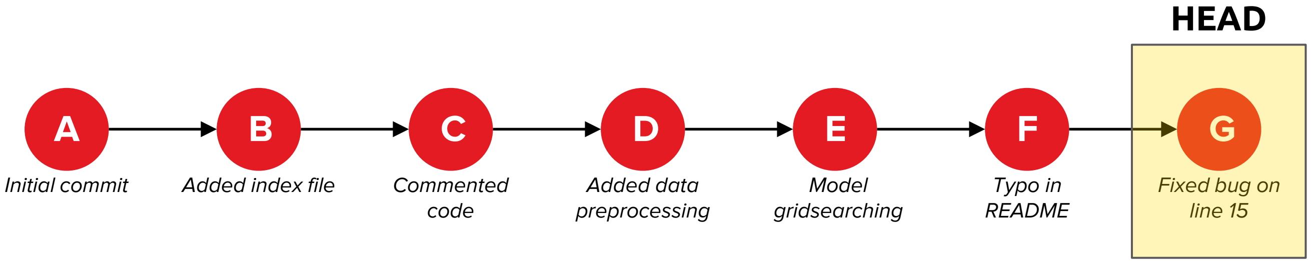
`git add .`	Add changes to staging area
`git commit -m ‘msg’`	Commit changes permanently
`git push`	Push committed changes to Github

`git status`	Ask Git what’s going on. You can do this anytime, and should do it often!

Now you:
Make another change and push it to your repo!

Git is a timeline of nodes

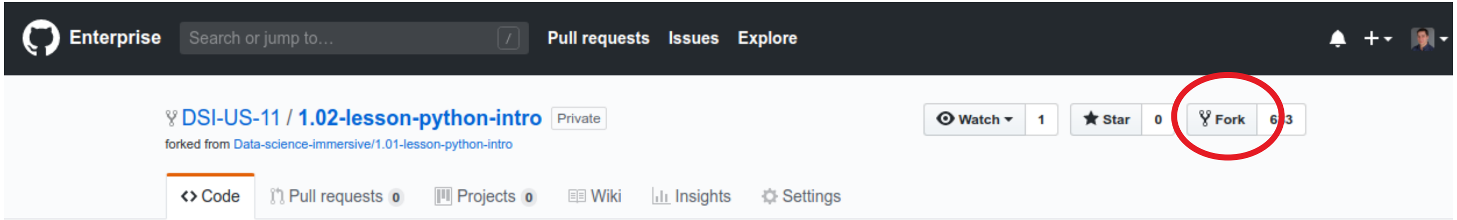
Why do we have to go through all these steps? It's so we can **time travel** if we make a mistake. In more advanced workflows, we can even **branch off** from our timeline to make different versions of our project.



Git for DSI Lessons

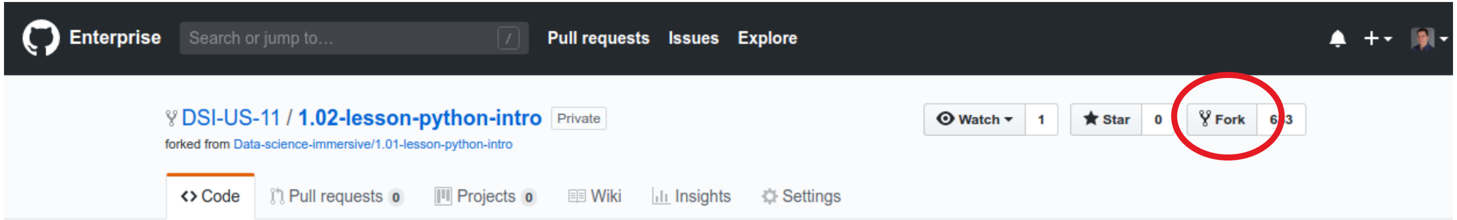
Where do lessons come from?

In our course, we also use Github as a **course management system**. Our lessons are Git repos, and you'll **fork** and **clone** them before each lesson.



Where do lessons come from?

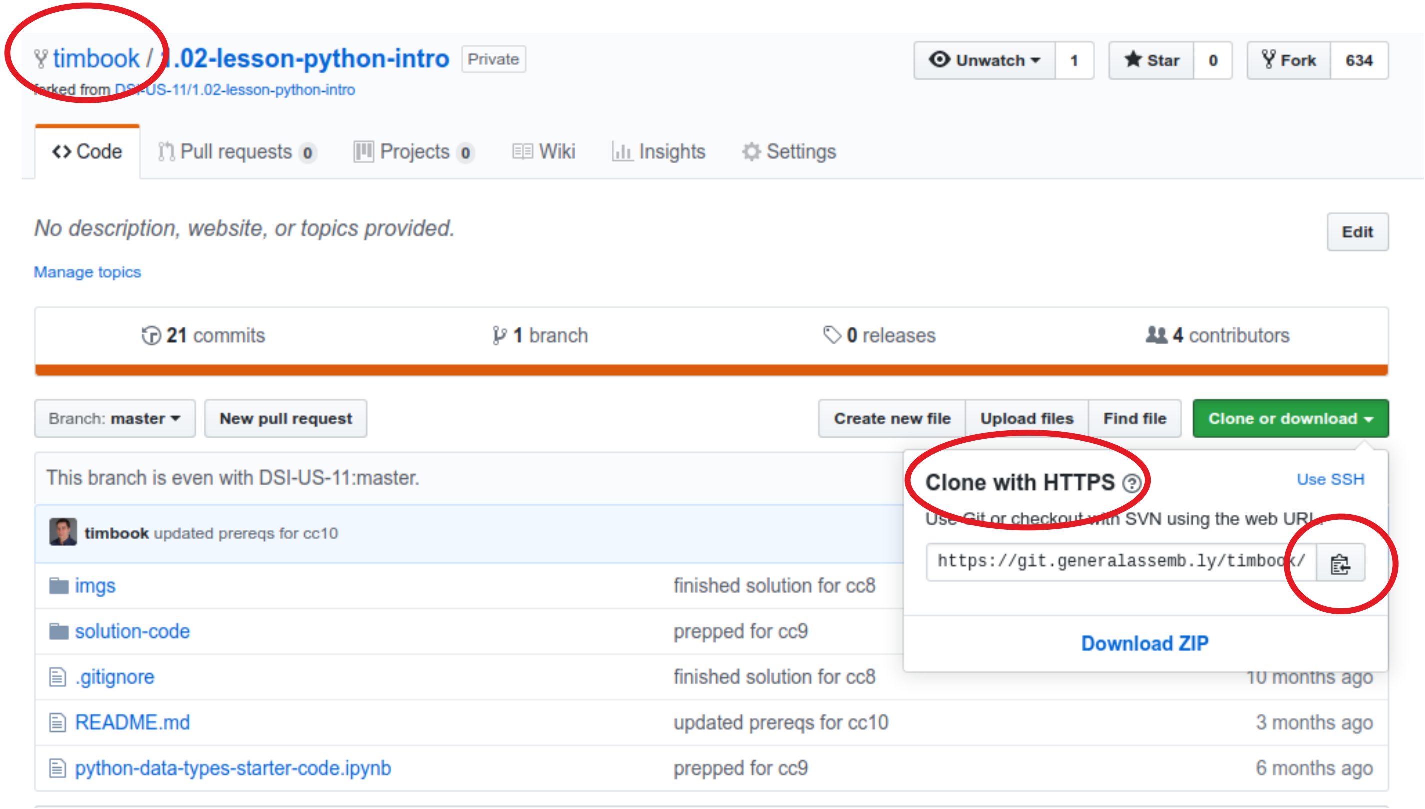
We do this so that I (the instructor) can push the work we did in class so you can see it later, while also maintaining your own private versions backed up on Github Enterprise.



DSI-US-11 / 1.02-lesson-python-intro Private

forked from Data-science-immersive/1.01-lesson-python-intro

Make sure you clone *your* repo!



timbook / 1.02-lesson-python-intro Private

forked from DSI-US-11/1.02-lesson-python-intro

No description, website, or topics provided.

This branch is even with DSI-US-11:master.

timbook updated prereqs for cc10

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

`https://git.generalassemb.ly/timbook/`

imgs	finished solution for cc8	
solution-code	prepped for cc9	
.gitignore	finished solution for cc8	10 months ago
README.md	updated prereqs for cc10	3 months ago
python-data-types-starter-code.ipynb	prepped for cc9	6 months ago

The Most Important Part!

Together, let's do the following:

1. **Fork** tomorrow morning's lesson
2. Create a personal GA folder and **clone** our lesson into it
3. Navigate into it and open our Jupyter Notebook
4. Run one line of code

Summary

What did we do today?

- Learned some basic command line
  - Movin' with **cd**
  - Lookin' with ls and **pwd**
  - Makin' with **mkdir** and **touch**
- Git
  - The Git workflow
  - How to use Git to get our course material

Cheat Sheet

`cd /path/to/folder`	Change directory (move)
`pwd`	Print working directory (where am I now?)
`ls`	List everything where I am
`mkdir folder-name`	Make a folder called **folder-name**
`touch file-name.py`	Make an empty text file called **file-name.py**
`echo string`	Sends **string** to stdout.

Git Workflow

`git add .`	Add changes to staging area
`git commit -m ‘msg’`	Commit changes permanently
`git push`	Push committed changes to Github

`git status`	Ask Git what’s going on. You can do this anytime, and should do it often!