Lab #8

Git

Agenda

- Final Project Info
- All things Git
- Make sure to come to lab for Python next week

Final Project Low Down

- The Projects are Creative AI, Arduino, Web Scheduler, iOS and Connect 4

- Notes about working on a Team

Being on a Team: Tricks for Maximizing Efficiency

- Take meeting minutes
  - Write down who is supposed to do what when and what time you are going meet up next
  - hint: make a .txt file and push it to your repo before and after EVERY meeting
- Actually meet up in person, group chat does NOT suffice
- Set times and stick with them!

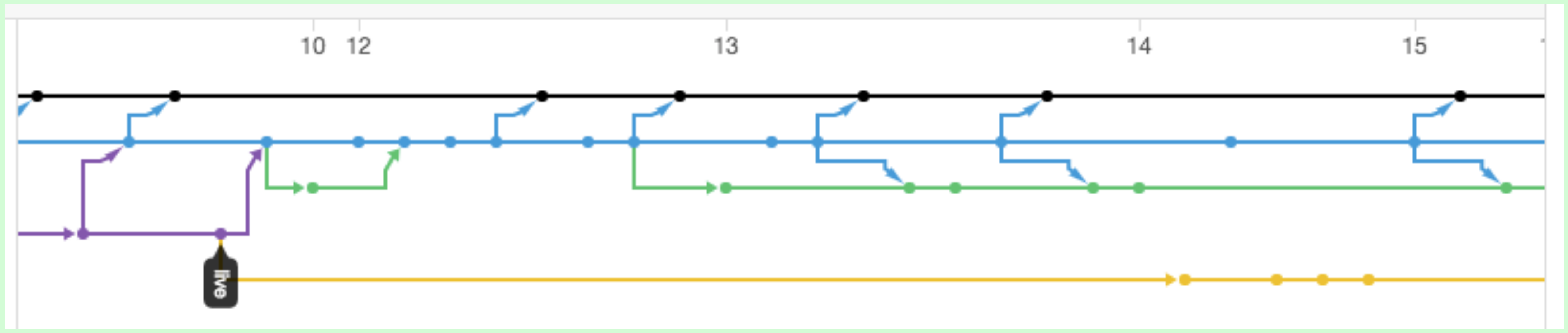
Using Git through the command line

Motivations behind version control

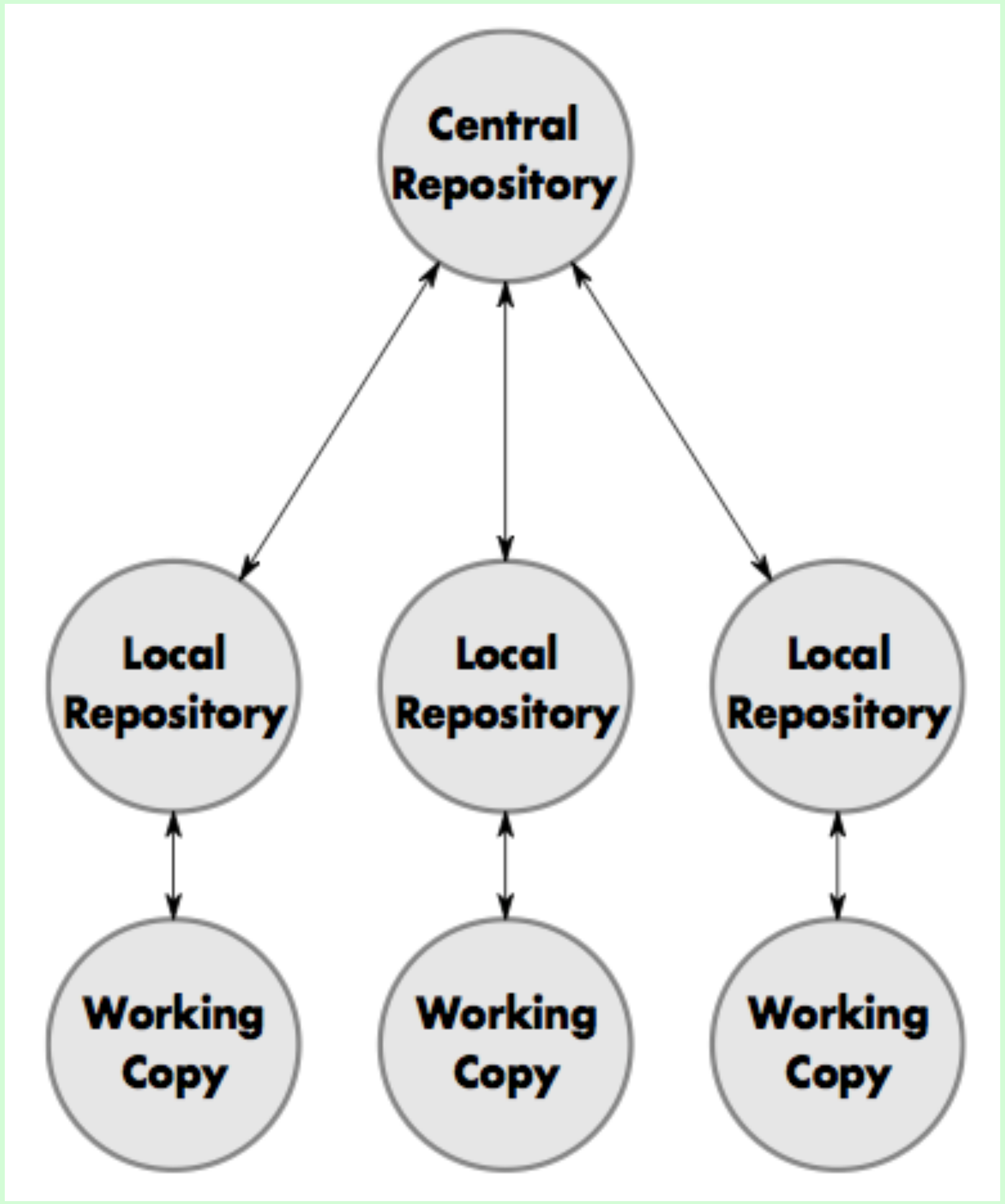
- Ease of collaboration & sharing
- Serves as a distributed backup

Motivations behind version control

- Ease of collaboration & sharing
- Serves as a distributed backup
- Builds a *narrative* of your project:

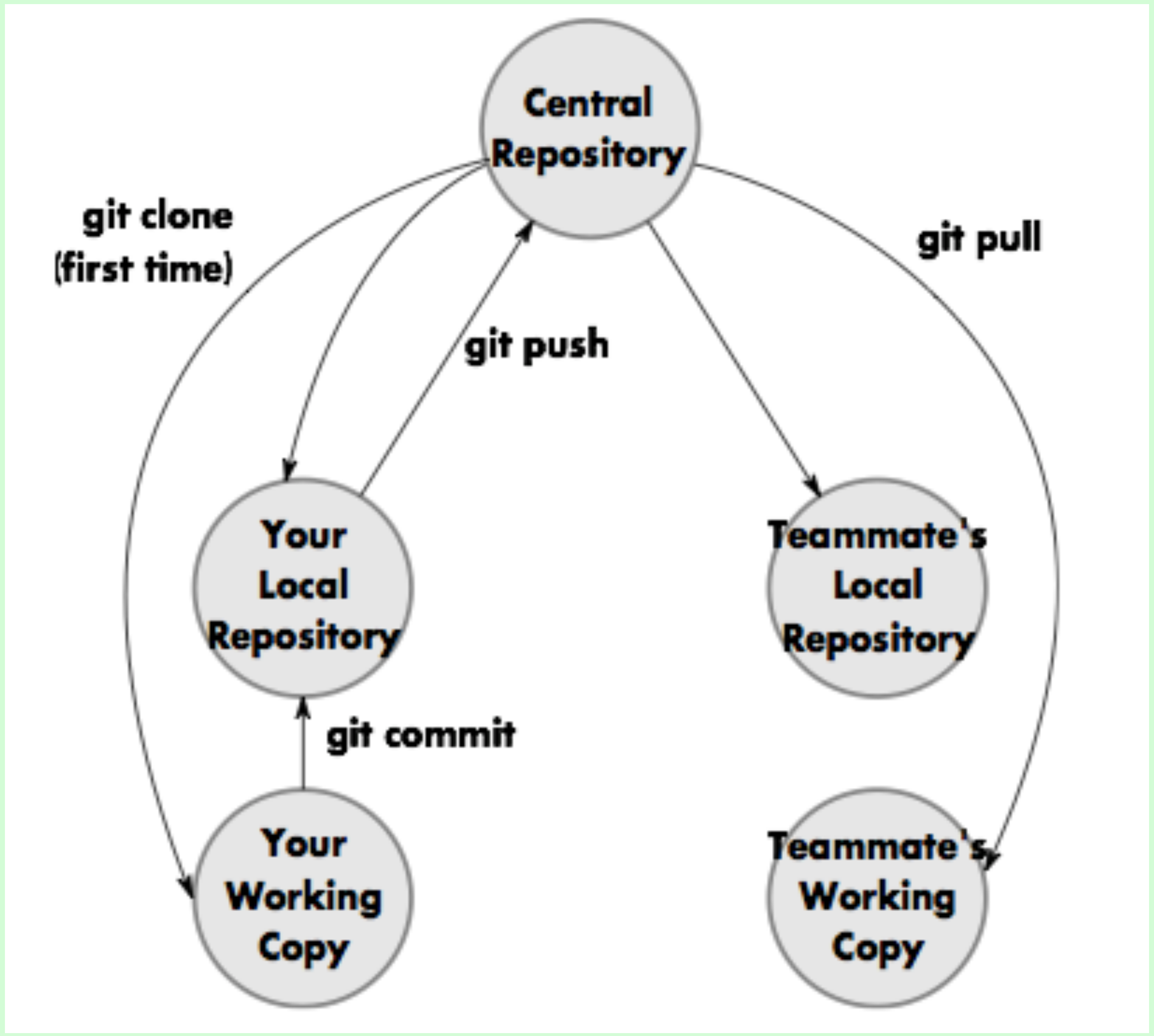


Distributed



Distributed repos (à la git)

Collaborating via git

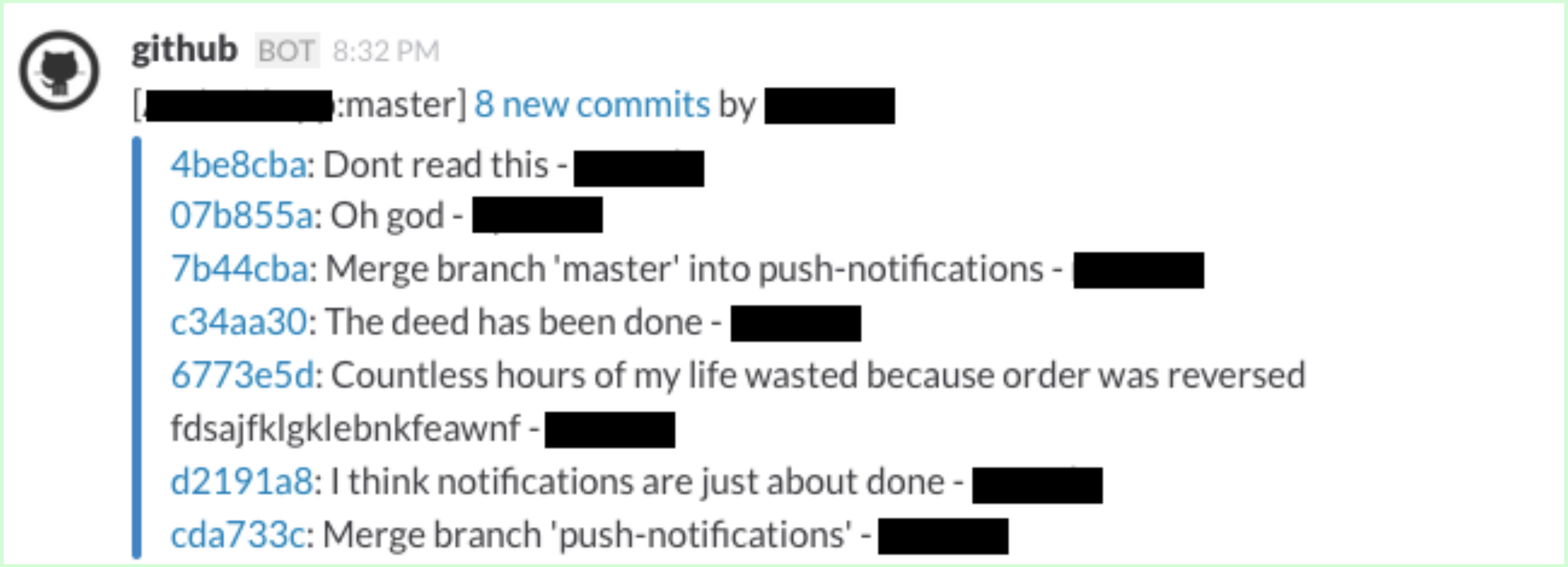


Best practices with git

- Don’t use commits as a way to “save” your code.
- Don’t wait too long to commit.
- Careful not to commit sensitive info when pushing to GitHub.
- Choose helpful commit messages. (Spoiler: good luck with this one at 4 AM.)

Best practices with git

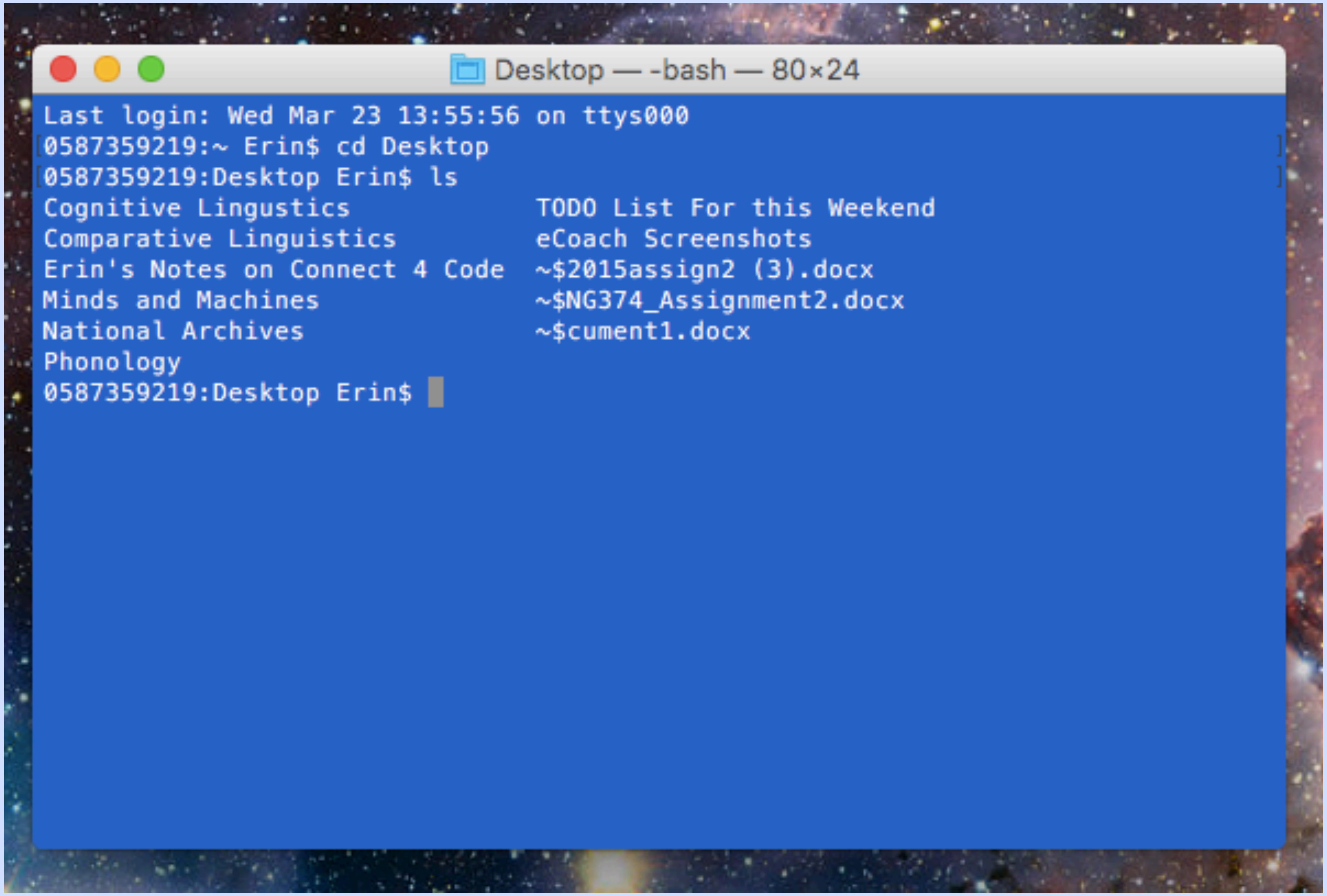
- Choose helpful commit messages. (Spoiler: good luck with this one at 4 AM.)



github BOT 8:32 PM
[[redacted]:master] 8 new commits by [redacted]

4be8cba: Dont read this - [redacted]
07b855a: Oh god - [redacted]
7b44cba: Merge branch 'master' into push-notifications - [redacted]
c34aa30: The deed has been done - [redacted]
6773e5d: Countless hours of my life wasted because order was reversed
fdsajfklgklebnkfeawnf - [redacted]
d2191a8: I think notifications are just about done - [redacted]
cda733c: Merge branch 'push-notifications' - [redacted]

Using Terminal



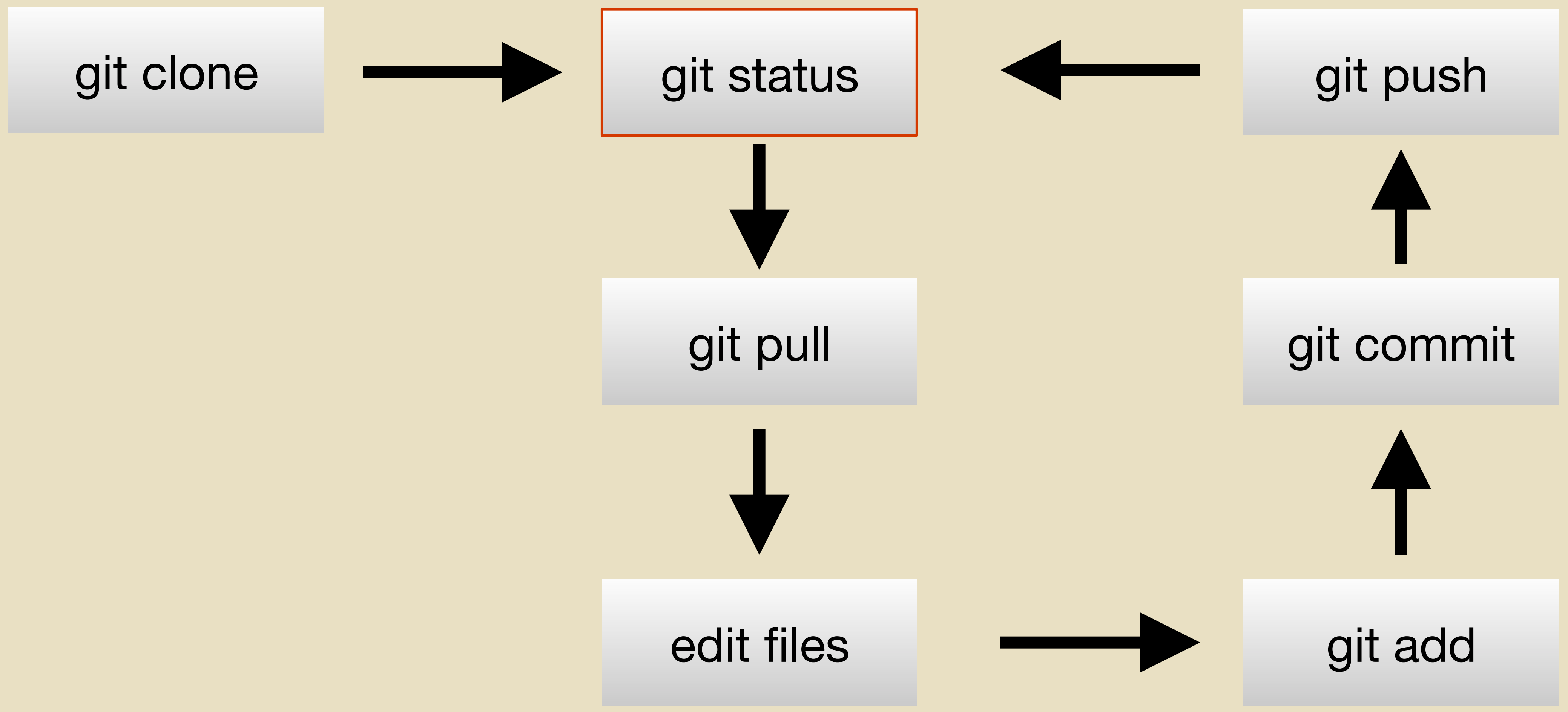
```
Last login: Wed Mar 23 13:55:56 on ttys000
0587359219:~ Erin$ cd Desktop
0587359219:Desktop Erin$ ls
Cognitive Lingustics            TODO List For this Weekend
Comparative Linguistics         eCoach Screenshots
Erin's Notes on Connect 4 Code  ~$2015assign2 (3).docx
Minds and Machines              ~$NG374_Assignment2.docx
National Archives               ~$cument1.docx
Phonology
0587359219:Desktop Erin$
```

Useful Commands To Know

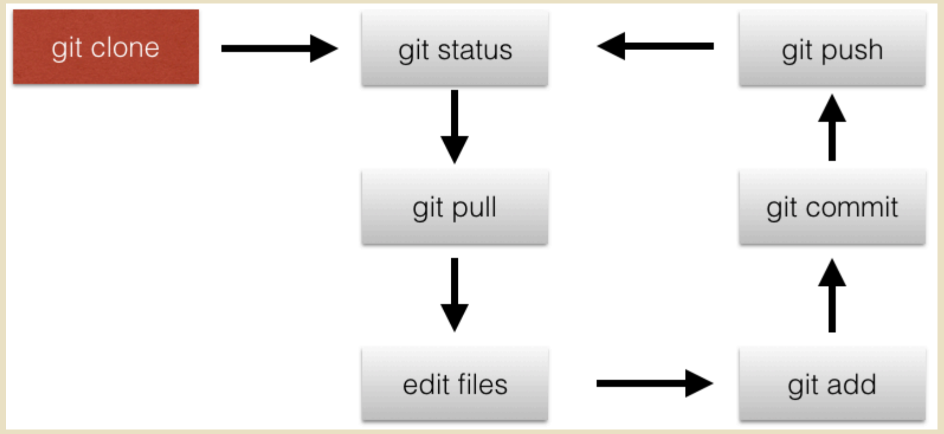
cd [path]	**c**hange **d**irectory to [path/place]	
ls	**l**i**s**t files/folders in directory	ls -la (detailed list)
pwd	**p**rint **w**orking **d**irectory	
q	**q**uit	
mkdir [folder name or path]	**m**a**k**e **dir**ectory	

Ctrl-A	Cursor to front of command	
Ctrl-E	Cursor to end of command	
Ctrl-L	Clear screen	
Ctrl-R	Reverse command search	Find that one long command you typed a few minutes ago

Getting Started



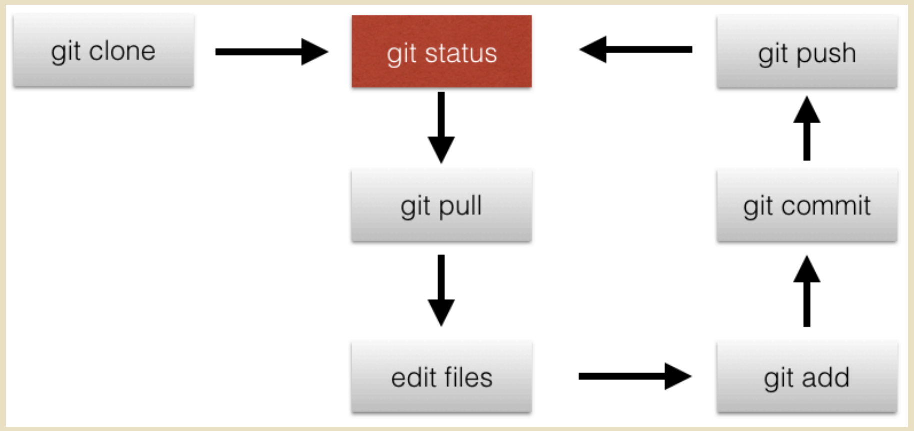
git clone



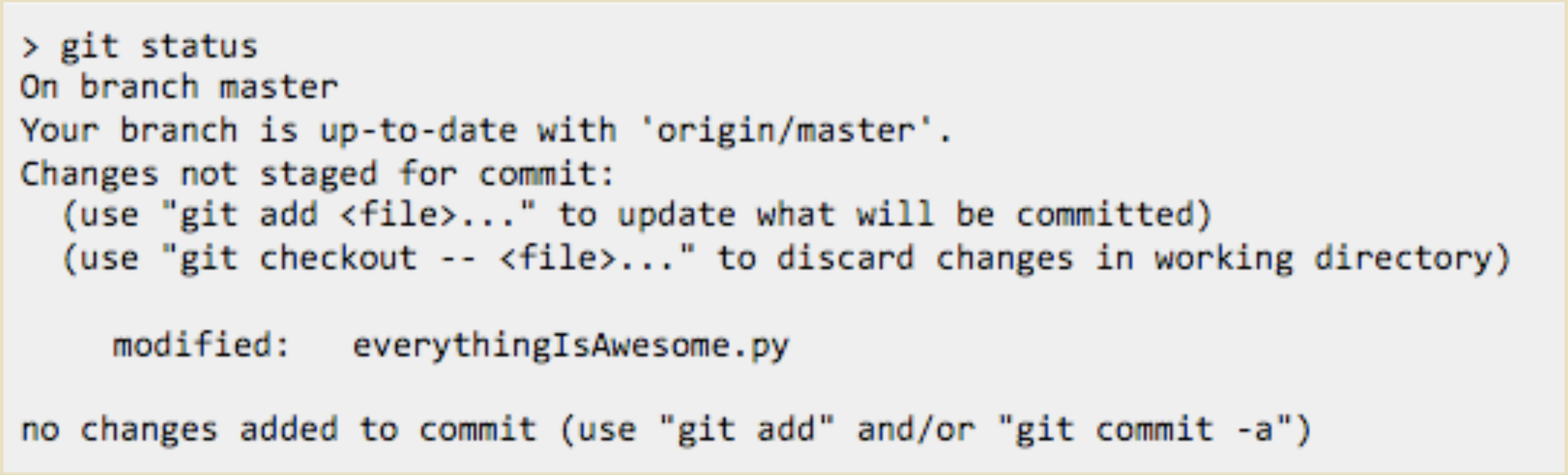
- This command makes a copy of a Git repository on your computer, in the directory that you run the command from
    - Example

        -
          ```
          > git clone https://github.com/yourName/hello.git
          ```
- If you ran this command from your Desktop, you would have a copy of the https://github.com/yourName/hello repository called `hello` on your Desktop
- Your `hello` repo on your Desktop is not the same copy as the repo you cloned from GitHub
- However, the other Git commands allow you to sync changes between the GitHub repository and your own

git status

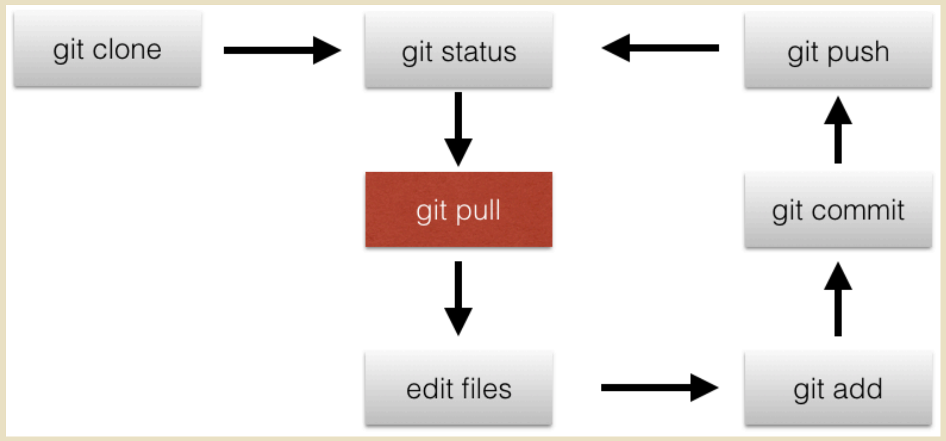


- This command will output some information on the "status" of the repository you're working in



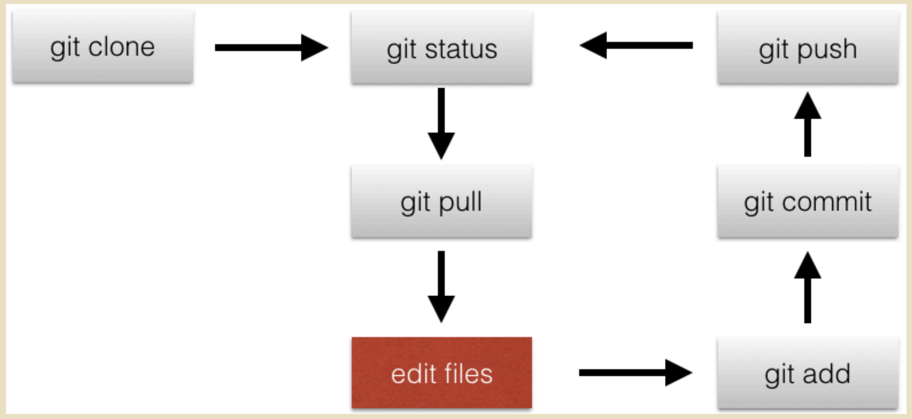
```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

     modified:   everythingIsAwesome.py

no changes added to commit (use "git add" and/or "git commit -a")
```

git pull



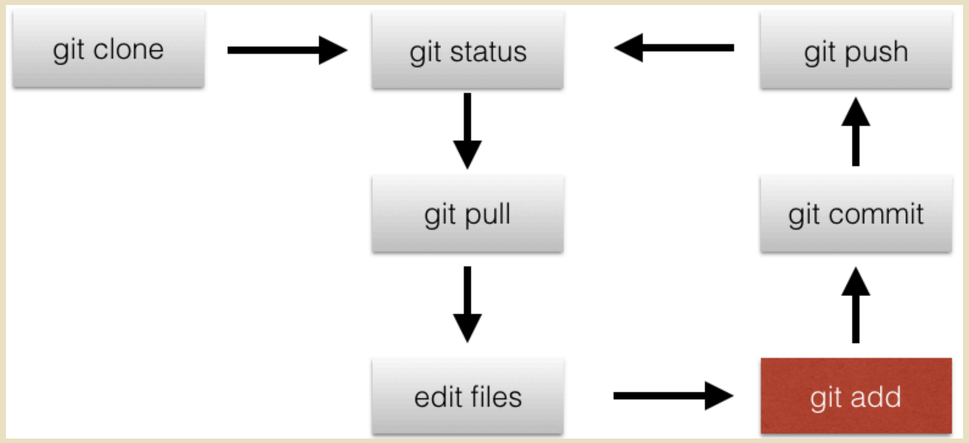
- This command "pulls" any changes from the centralized repository you cloned to your computer
- If someone else changed the repository that you cloned from, you need to be able to get the latest version of that repository
- `git pull` will make your local copy of the code up to date with whatever changes are in the repository you cloned from

edit files



- Just like you would in whichever IDE that you are using!

git add



- Going back to the idea of version control: you take “snapshots” of your code at different points in its lifetime

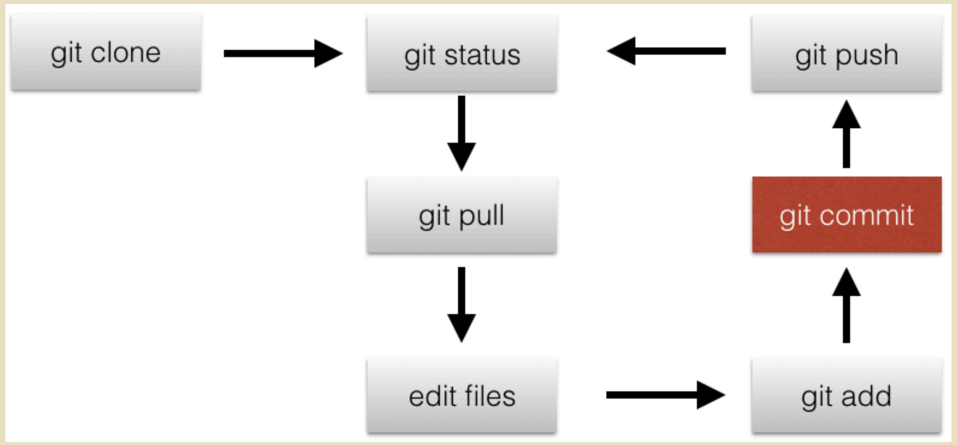
- `git add` is the first step to taking those “snapshots”

- It makes Git aware that you changed some files

- Say you edited `myFirstGitProgram.cpp` in your `hello` repository. You want to include your changes to `myFirstGitProgram.cpp` in your next “snapshot” of your code:

```
> git add myFirstGitProgram.cpp
```

git commit

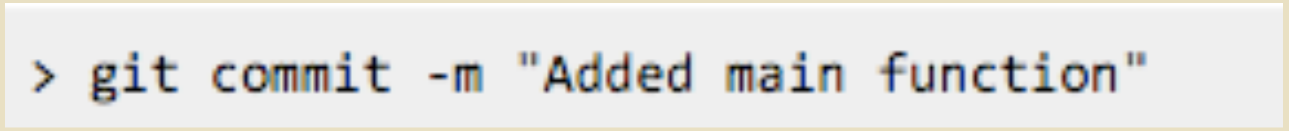


- This command “takes a snapshot” of your local repository, by saving the state of whatever files you added

- `git commit` allows you to make versions of your code

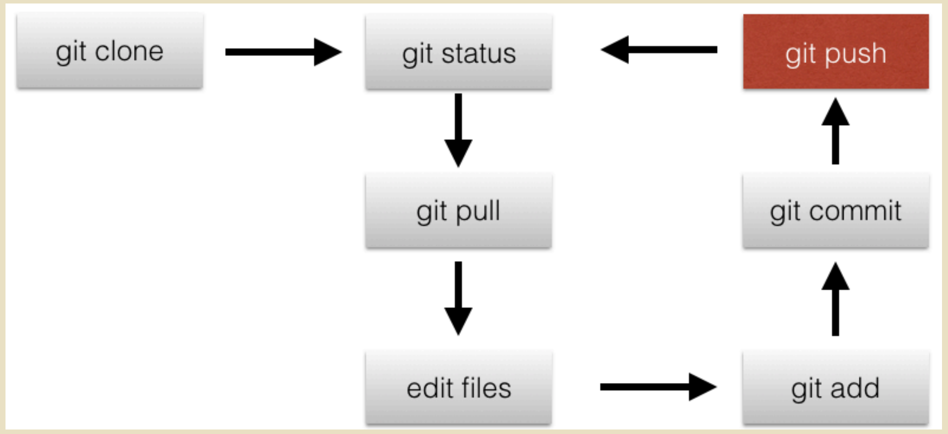
- **You can only commit files that you have first added using `git add`**

- When you commit your code, you must always include a **commit message** that explains briefly what updates you have made to the code



```
> git commit -m "Added main function"
```

git push



- This command sends your local version of a repository back to the centralized repository that you cloned from

- Say you cloned from a repo on GitHub and committed some changes

- `git push` sends the changes you committed *back to the centralized repository hosted on GitHub*

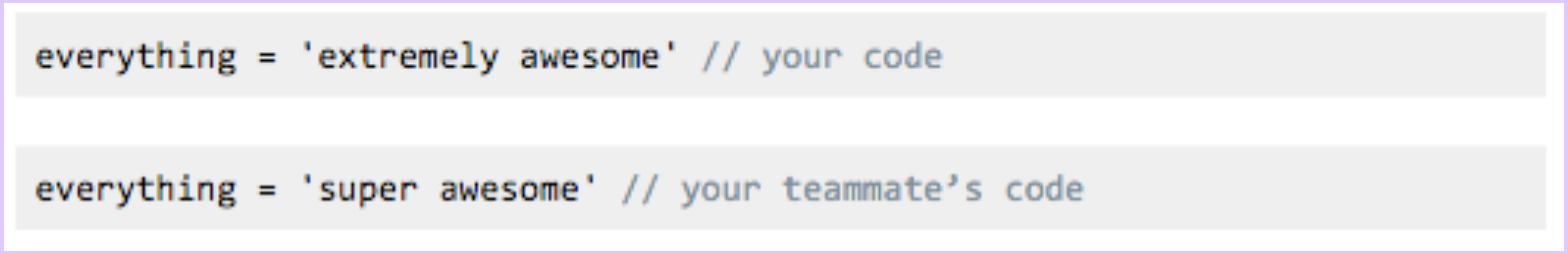
```
> git push
```

Potential Problems in git

Merge Conflicts

- Let’s say you and your teammate **both edited line 1** of `myFirstGitProgram.cpp`. Your teammate pushed her changes to the centralized repository first, and then you committed your changes and pulled from your centralized repository

- Now Git is confused because there are two versions of the same code, and Git doesn’t know which one is right

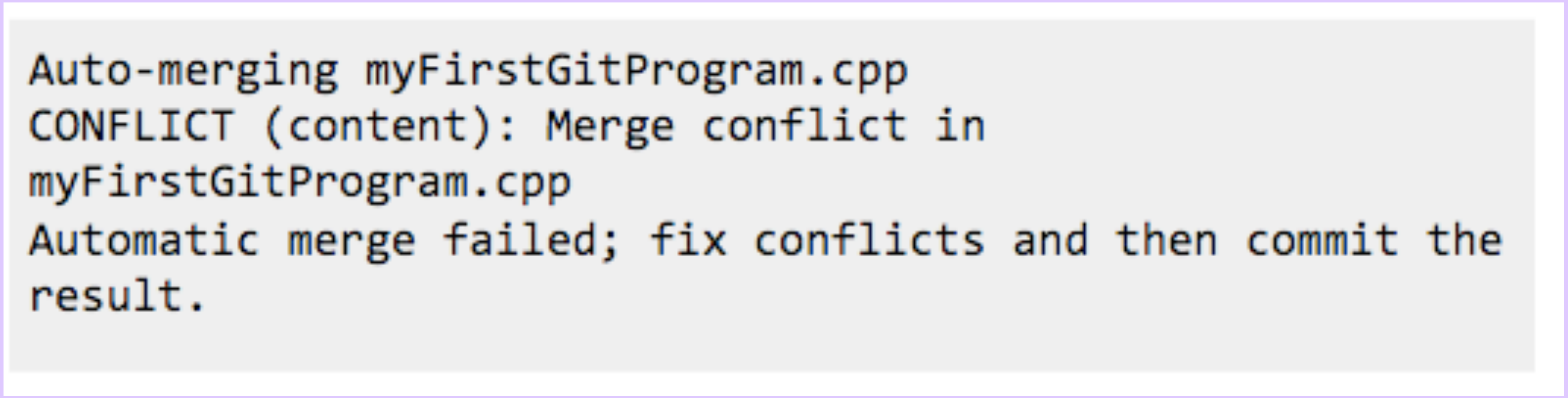


```
everything = 'extremely awesome' // your code
```

```
everything = 'super awesome' // your teammate’s code
```

Merge Conflicts

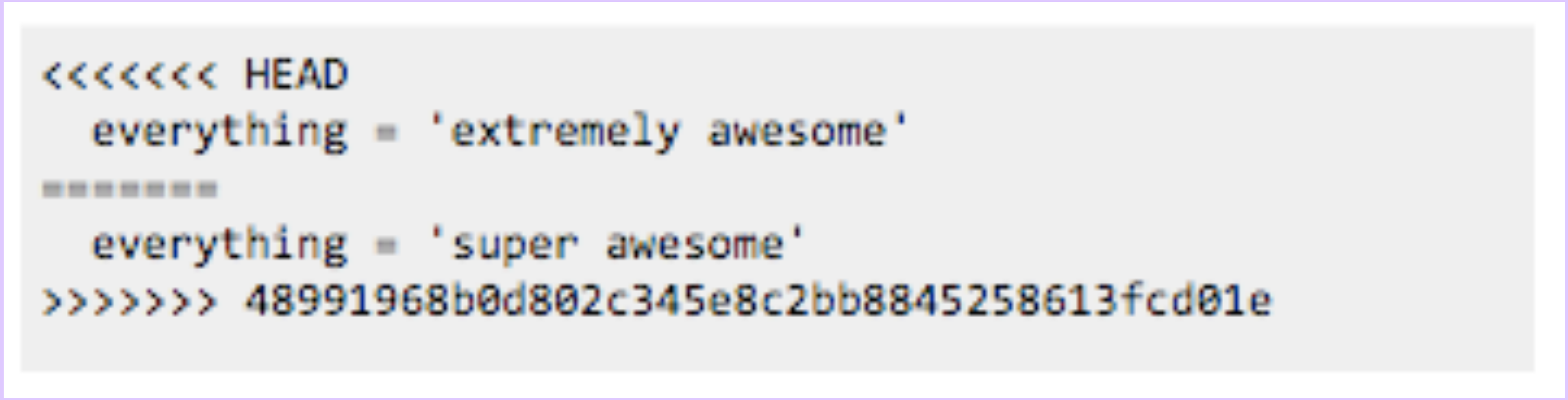
- That is called a merge conflict
- When it happens you will see something like this in the terminal:



```
Auto-merging myFirstGitProgram.cpp
CONFLICT (content): Merge conflict in
myFirstGitProgram.cpp
Automatic merge failed; fix conflicts and then commit the
result.
```

Merge Conflicts

- Your `myFirstGitProgram.cpp` file will look like something like this



```
<<<<<<< HEAD
  everything = 'extremely awesome'
=======
  everything = 'super awesome'
>>>>>>> 48991968b0d802c345e8c2bb8845258613fcd01e
```

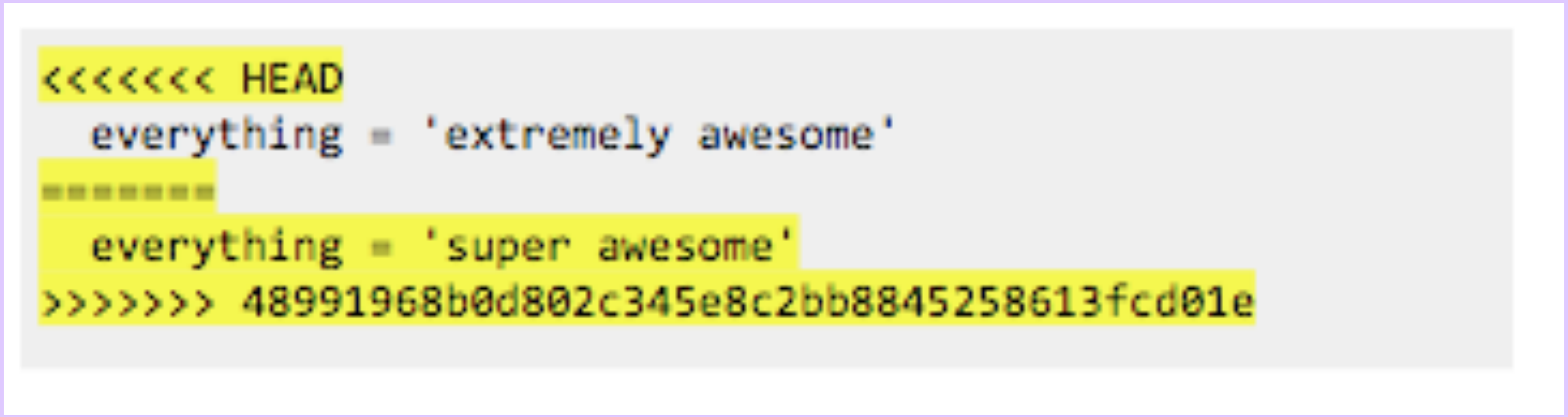
- Don’t be scared by these symbols! Git just puts them there to differentiate between the two versions of the code it’s looking at

- **The top part above ===== is your version of the code, the bottom part is the version that you pulled (your teammate’s version)**

Merge Conflicts

- To fix a merge conflict, delete all the symbols Git added along with the version of the code you don't want to keep

- In this example, everything highlighted in yellow will be deleted



```
<<<<<<< HEAD
  everything = 'extremely awesome'
=======
  everything = 'super awesome'
>>>>>>> 48991968b0d802c345e8c2bb8845258613fcd01e
```

Merge Conflicts

- `git add` the file after you delete the symbols, and `git commit` to “resolve” the merge conflict

- That’s all you need to do to fix a merge conflict !!!

fatal: not a Git Repository

- If you ever get this error when running a Git command, you’re likely in the wrong directory
- Type `pwd` to print the directory that you’re currently in
- Navigate to where you should be (your directory that has your Git repo)

Lab

- Two parts for the lab today.
  - No exam practice
- Practice learning Git
  - Complete “Learn Git in 15 minutes” tutorial
  - Submit Google form answering questions on Git
- Also opportunity to ask questions about your team GitHub repository!