

## Reminders

### Upcoming due dates

Mon Oct 6th Quiz 1

Wed Oct 9th Pre-course survey

Fri Oct 10th Discussion Lab 1

#FinAid quiz on Canvas

Discussion section this week is a tour of Datahub and an ungraded Python review notebook

# Version Control

## Data Science in Practice

# This sucks

archived version of my Documents folder from ~ 2012

📄 LNAI_fulltext.pdf
📄 LNAI_v4520.pdf
▶ 📁 Neuromorphic BBD book
▶ 📁 Neuromorphic BBD book - from Desktop
▶ 📁 Neuromorphic BBD book -- FINAL version 99% sure
📄 neuromorphic book abstract final.pages
📄 neuromorphic robots book abstract v1.pages
📄 NeuromorphicBookChapter2011.pdf
📄 neuroreport_v18_n17_2007.pdf
📄 pnas_v104_n9_pp3556-3561.pdf
📄 robotics and automation magazine (conflict at 2012-07-28_00-23-24)
▶ 📁 Robotics and Automation Magazine 2009 final version
▶ 📁 Robotics and Automation Magazine 2009 f *** ed up copy due to sync with laptop i think
▶ 📁 Robotics and Automation Magazine 2009 may be jacked tex file
▶ 📁 Robotics and Automation Magazine 2009 not final version, too many refs
▶ 📁 Robotics and Automation Magazine 2009 not the final, too many refs to be it
📄 Robotics and Automation Magazine 2009.zip
📄 robotics and automation magazine.pages
▶ 📁 Rome JIN Submission
📄 SegwaySoccerICRA2006.pdf

Several months after finishing a writing project, I wanted to keep only the final version of the many different revisions... figuring out which one was the version actually sent to the publisher was hard!

# A step in the right direction

Among the unique items consumed, only a few may be consumed regularly. To assess this, we asked which f/b were consumed by at least 100 people (~0.5% of the cohort) for  $\geq 7$  of 14 days. We found that 19,445 users consumed at least 1 item  $\geq 7$  days (mean 3.85 items,  $sd$  2.96), however, only 108 items (75 foods and 33 beverages) were consumed by at least 100 people for  $\geq 7$  days (**Fig 4c, Table S10**). For example, the two most consistent items, coffee and tea, were consumed by 10,282 and 2,516 users for  $\geq 7$  days, respectively. These and the majority of other habitual items, more commonly consumed in the morning, suggest that foods and beverages with high habitual recurrence are more likely to occur earlier in the day, implying more routine food choices associated with morning consumption. No food was consumed by at least 100 people for all 14 days and only 4 beverages were consumed by at least 100 people for all 14 days. Coffee was logged by 1534 users for all 14 days; milk, black coffee, and tea are the other 3 beverages consumed by at least 100 users for 14 days.

Despite the diversity of foods logged, a small subset of items typically accounts for a large share of an individual's intake. We found 50% of users (out of the 21k) (1) reach 50% logging (half their total food/beverage items) with 9 unique items, (2) 75% consumption with 21 unique items, (3) 90% consumption with 35 items (**Fig 4d and Table S4**). On average, a user's most frequently consumed item is logged 16.23 times, yet only about half a user's unique items (mean 25.66,  $sd$  11.23) are consumed more than once, with single-consumption (novel) items making up 48% of their total diversity on average. Dense ranking reveals that any items ranked beyond 22nd in popularity are consumed exactly once ( $sd$  0) (**Fig 4e**). Frequency distribution of time of consumption

Tyler Tran  
Deleted: >

Tyler Tran  
Deleted: std

Tyler Tran  
Deleted: For example, coffee is consu  
users for  $\geq 7$  days and tea is consume  
are the top 2). These foods were also  
among items largely consumed durin  
day. This may be because these are f/  
consumed in the morning are more li  
habitually than f/b consumed later in  
evening.

Fleischer, Jason  
Deleted: 0.5%

Fleischer, Jason  
Deleted: of the cohort

Tyler Tran  
Deleted: Many f/b may constitute hal  
consumed more frequently than othe

Tyler Tran  
Deleted: A user's most frequently cor  
logged on average 16.23 times. Howe

Tyler Tran  
Deleted: std

# Version Control

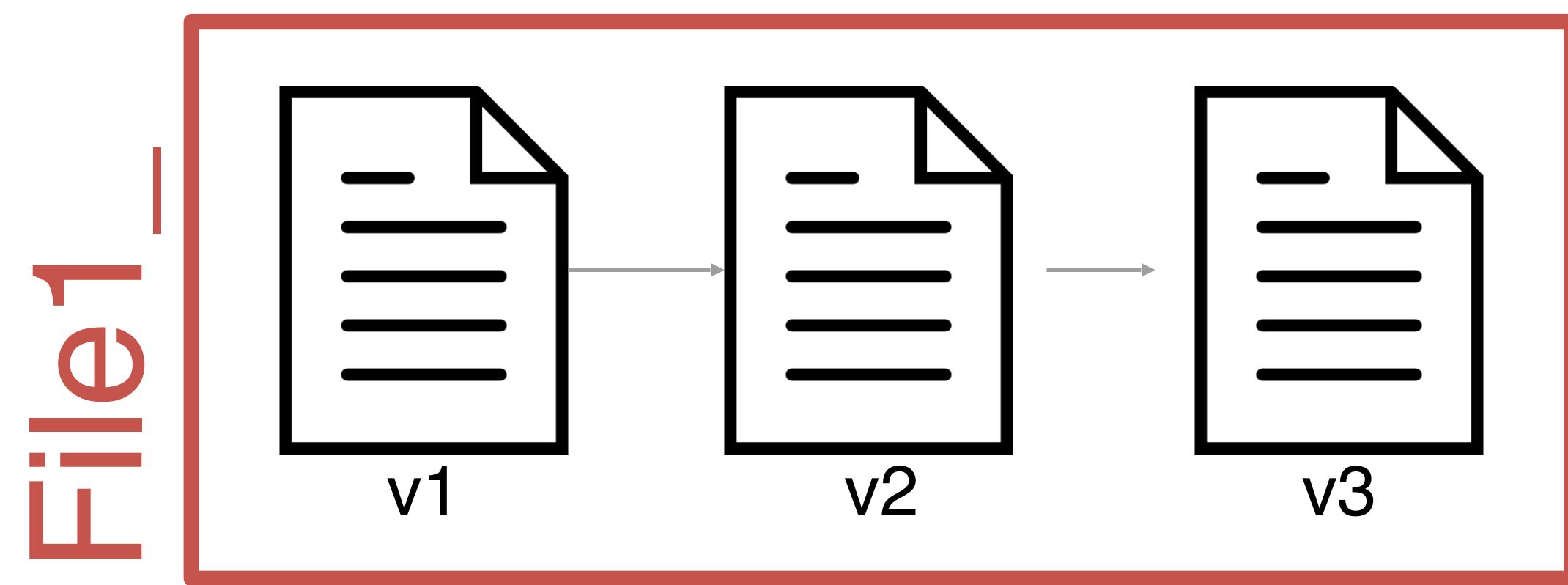
- Enables multiple people to simultaneously work on a single project.
- Each person edits their own copy of the files and chooses when to share those changes with the rest of the team.
- Thus, temporary or partial edits by one person do not interfere with another person's work

What is version control?

A way to manage the evolution of a set of files

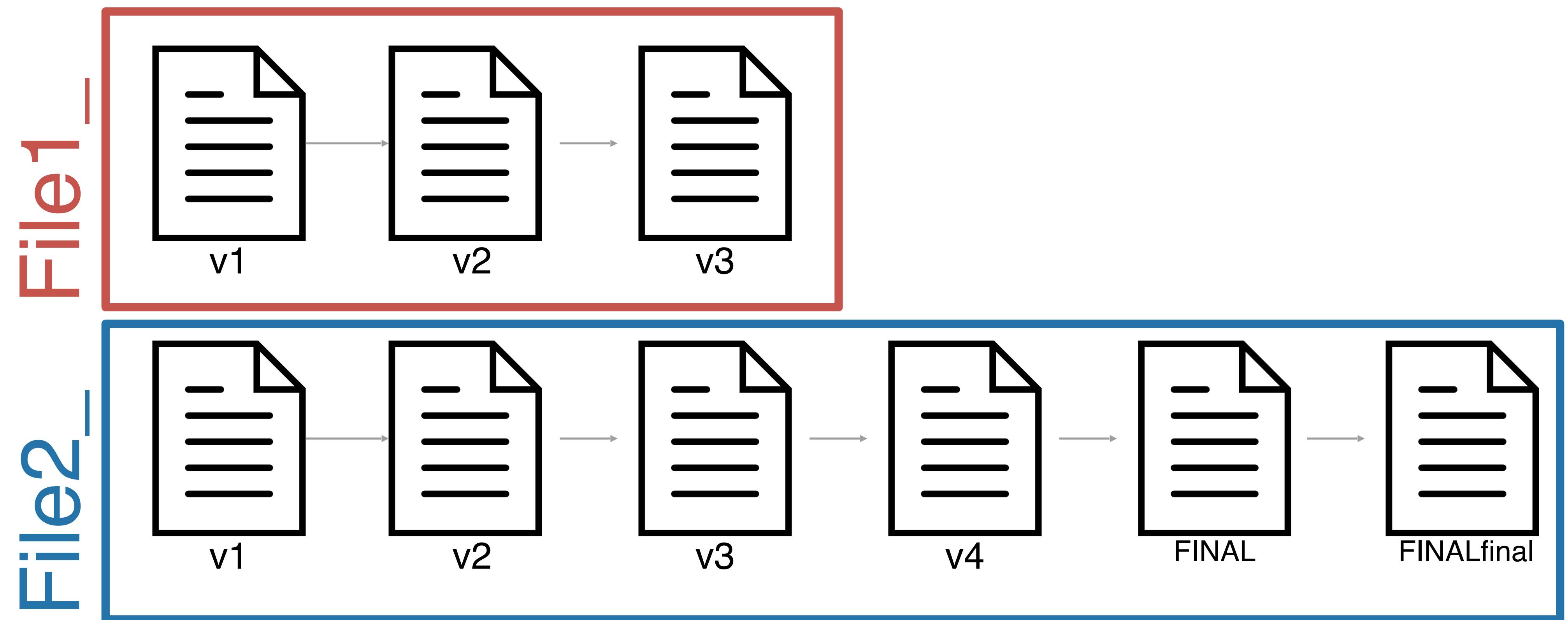
# What is version control?

A way to manage the evolution of a set of files



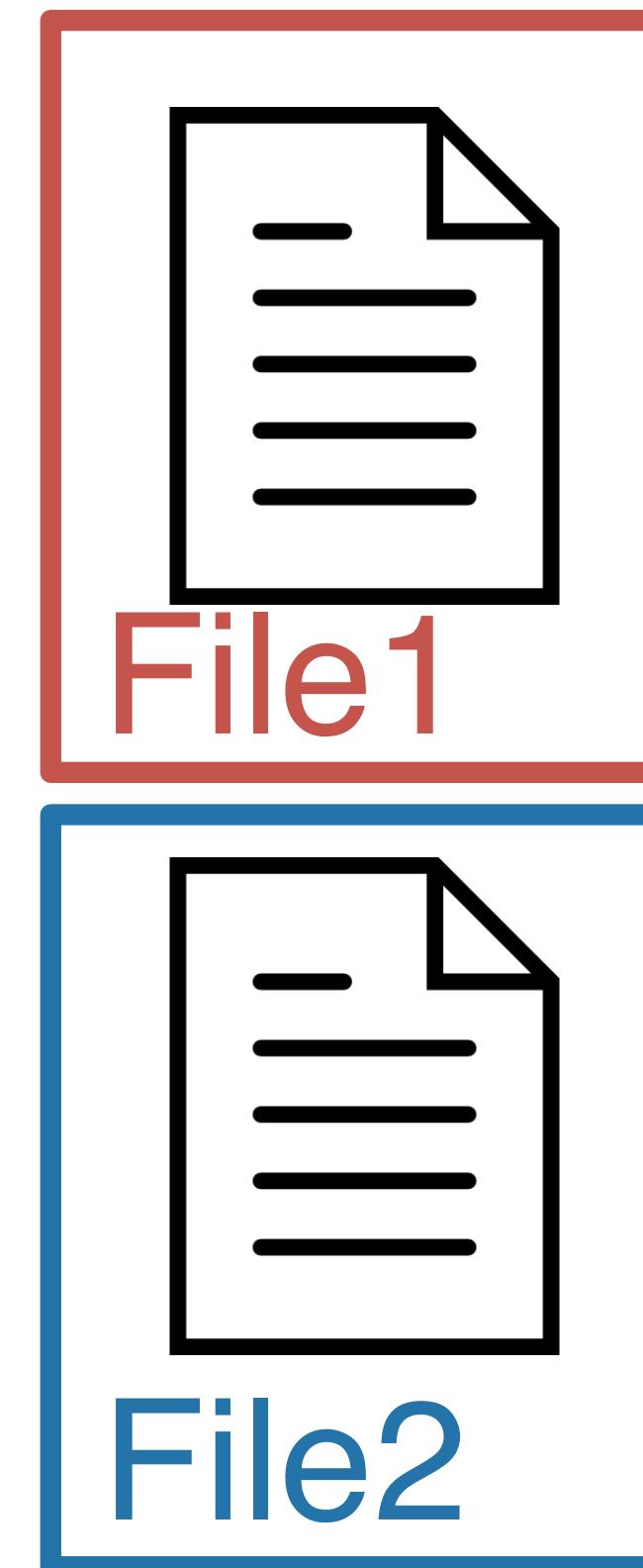
# What is version control?

## A way to manage the evolution of a set of files



# What is version control?

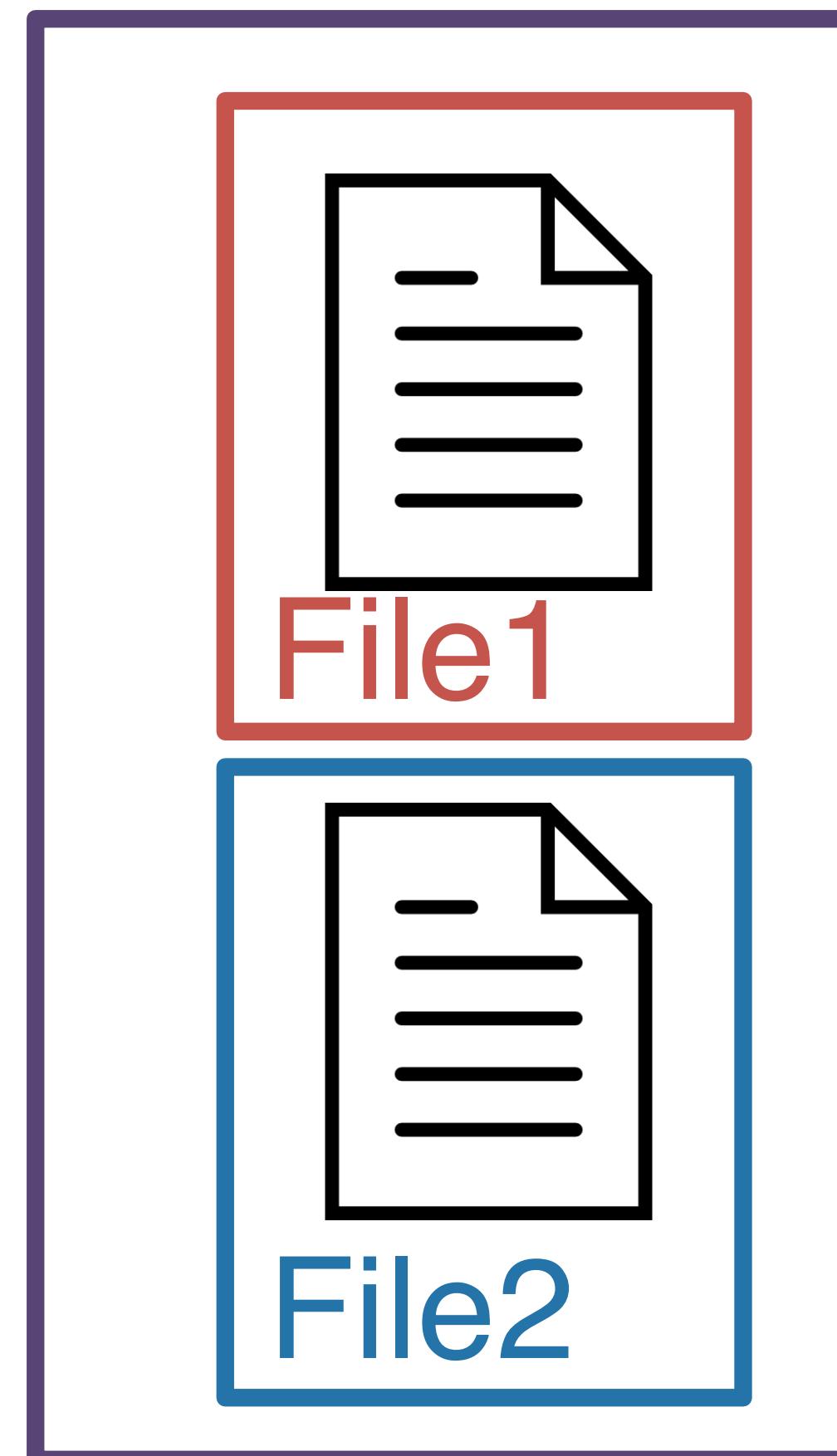
A way to manage the evolution of a set of files



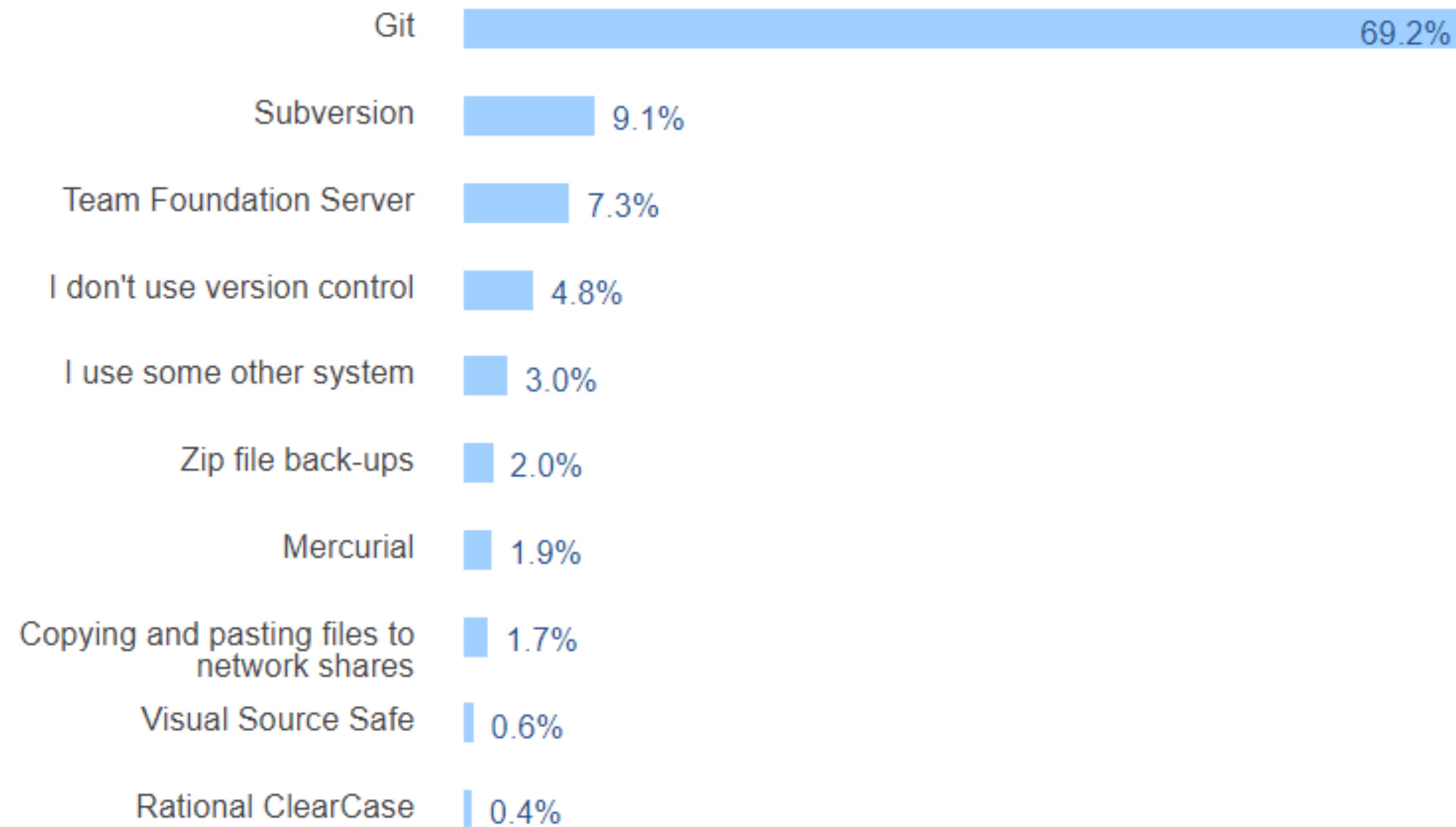
When using a version control system,  
you have **one copy of each file** and the  
*version control system tracks the  
changes* that have occurred over time

# What is version control?

A way to manage the evolution of a set of files



The set of files is referred to as a **repository (repo)**

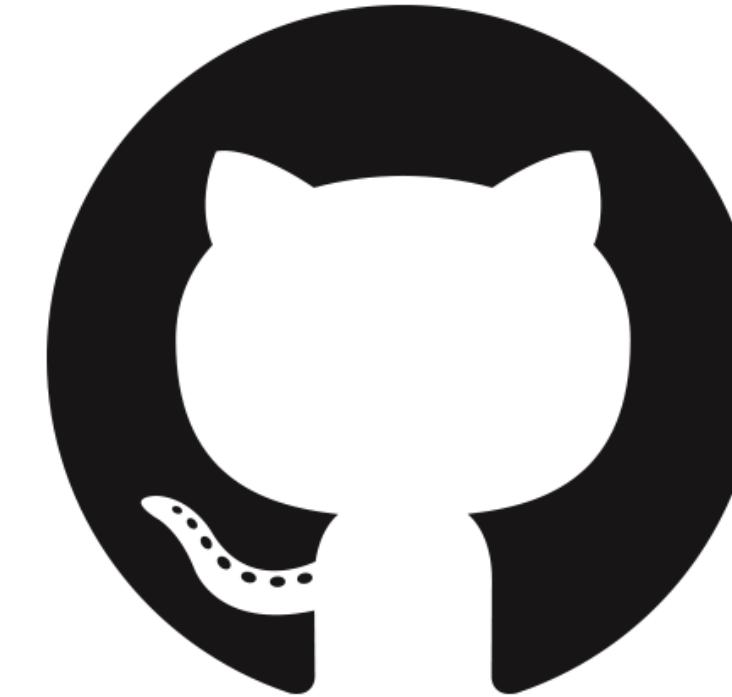


# git & GitHub

# git

the version control system

~ Track Changes  
from Microsoft  
Word....on  
steroids



**GitHub** (or Bitbucket or  
GitLab) is the home **where  
your git-based projects live**

on the Internet.

~ Dropbox +  
social media for  
programmers

# What version control looks like

```
$ git clone https://www.github.com/username/repo.git  
$ git pull  
  
[... edit some files, make changes ...]  
  
$ git add -A  
$ git commit -m "informative commit message"  
$ git push
```

Git on command line

# What version control looks like

```
jfleischer@dsmlp-jupyter-jfleischer:~/private/GitExercise_Con  
flicts$ git diff  
diff --git a/Project.ipynb b/Project.ipynb  
index d938ddb..3337e25 100644  
--- a/Project.ipynb  
+++ b/Project.ipynb  
@@ -13,10 +13,21 @@  
     },  
     {  
         "cell_type": "code",  
-        "execution_count": null,  
+        "execution_count": 4,  
         "id": "8d446506",  
         "metadata": {},  
-        "outputs": [],  
+        "outputs": [  
+            {  
+                "data": {  
+                    "image/png": "iVBORw0KGgoAAAANSUhEUgAAAiEAAAGHCAYAAABm  
-----
```

Git on command line

# What version control looks like

COGS108 / **GitExercise\_Conflicts**

Type  to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

**GitExercise\_Conflicts** Public

Edit Pins Watch 1 Fork 2 Star 0

main 2 Branches 0 Tags

Go to file + Code About

 **jasongfleischer** Clear all outputs instruction added 550d5ad · 2 hours ago 12 Commits

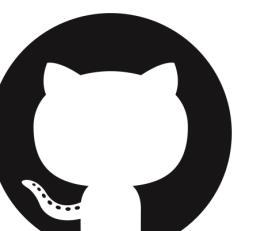
 .gitignore Initial commit 4 months ago

 LICENSE Initial commit 4 months ago

 Project.ipynb updated plot title on main 4 months ago

 README.md Clear all outputs instruction added 2 hours ago

 README MIT license



## GitExercise\_Conflicts

This exercise assumes you are using recent COGS 108 Datahub setup (which is Jupyterlab with the git extension built on nbdime). But you can do everything here using your own computer if you set it up with the same tools. If

No description, website, or topics provided.

Readme  
MIT license  
Activity  
Custom properties  
0 stars  
1 watching  
2 forks  
Report repository

Releases

No releases published  
[Create a new release](#)

# GitHub

# What version control looks like

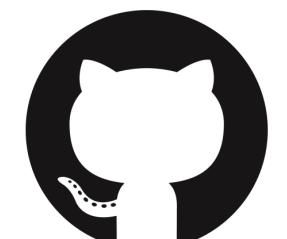
Filter files... 

1 file changed +43 -44 lines changed   Top  Search within code 

README.md 

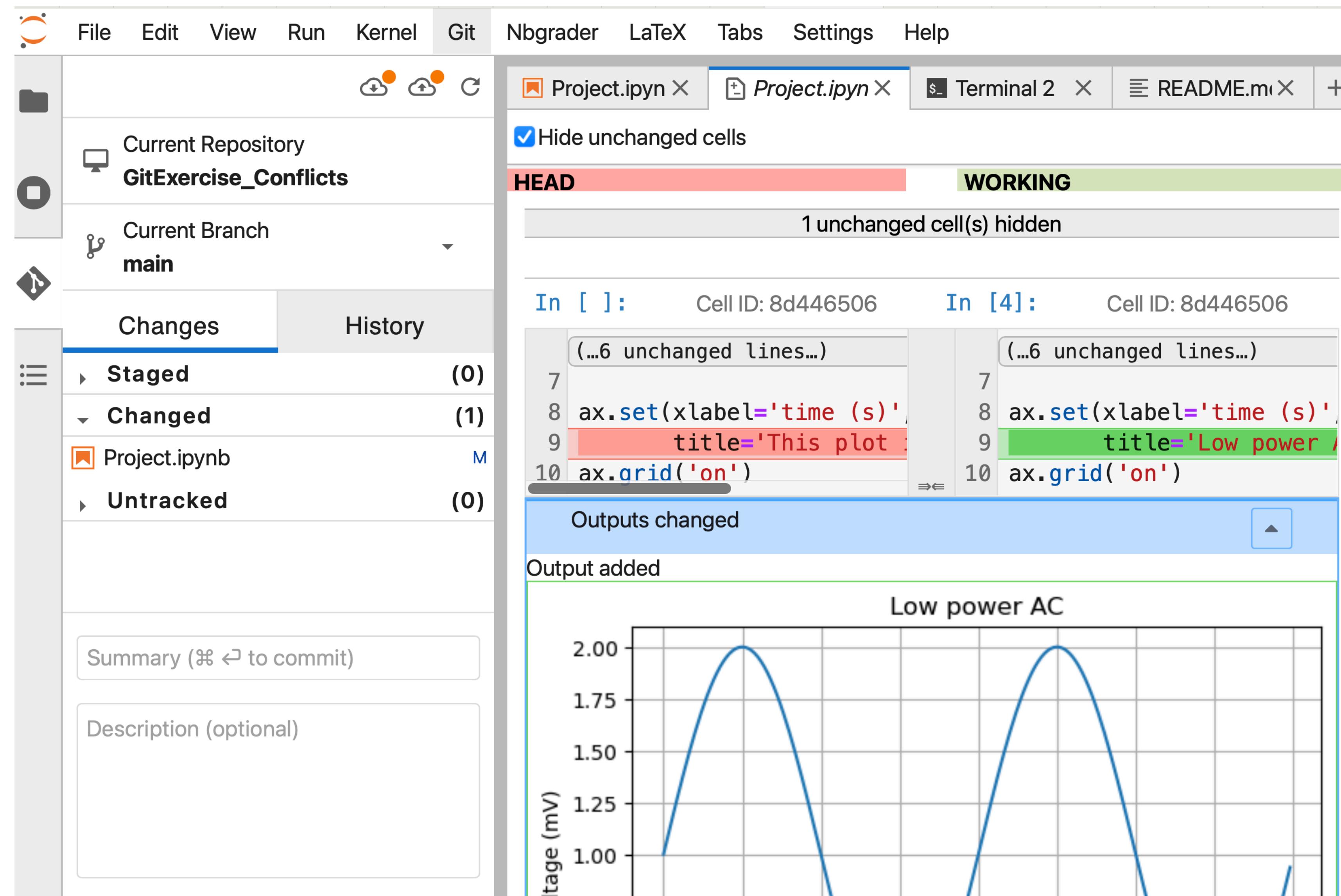
+43 -44       ...

	README.md	README.md
5	- This repo contains two branches: main and ex1. There is a single line text conflict in one cell between the two branches. The goal is to merge branch ex1 into branch main.	5 + This repo contains two branches: main and ex1. There is a single line text conflict in one cell between the two branches. The goal of this exercise is
6	-	6 + twofold; to learn how to stage and commit changes to a branch and then to learn how to merge the changes from one branch to another.
7	-	7 +
8	- 1. Clone this repo to your datahub or local computer. In a terminal window type `git clone <a href="https://github.com/COGS108/GitExercise_Conflicts.git">https://github.com/COGS108/GitExercise_Conflicts.git</a> `	8 + We will do two separate git operations
9	- 2. This clone only brings the main branch to your computer... we also need a local copy of the ex1 branch which you can get by typing the following two lines of code in the terminal (press return each time) 	9 + - add a new change to branch main, thus introducing a second difference with ex1 branch
10	- `cd GitExercise_Conflicts` 	10 + - then we will merge the changes from branch ex1 into main
11	- `git checkout -b ex1 origin/ex1` 	11 +
12	- that last line sets up a local branch called ex1 to track the remote branch of the same name (origin is git's word for the remote repo)	12 + Here are step-by-step instructions assuming that you have logged in to Datahub to do this:



GitHub

# What version control looks like



Datahub  
@ UCSD

<https://forms.gle/8UeUL2Ux4YtG2CVr8>

# Version Controller

How do you typically interact with git?

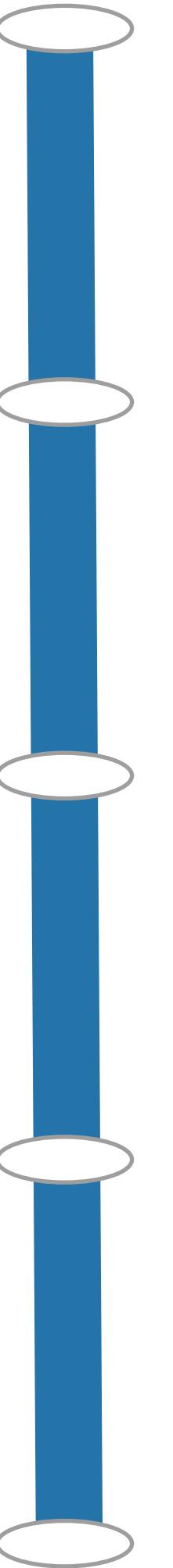
A. I don't

B. command line

C. GUI GitHub Desktop

D. GUI: SourceTree

E. GUI: other



<https://git-scm.com/downloads>



# git --distributed-even-if-your-workflow-isnt

Type / to search entire site...

**About**

**Documentation**

**Downloads**

GUI Clients  
Logos

**Community**

The entire [Pro Git book](#) written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

**Downloads**

 macOS     Windows  
 Linux/Unix

Older releases are available and the [Git source repository](#) is on GitHub.

**GUI Clients**

Git comes with built-in GUI tools ([git-gui](#), [gitk](#)), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

**Git via Git**

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

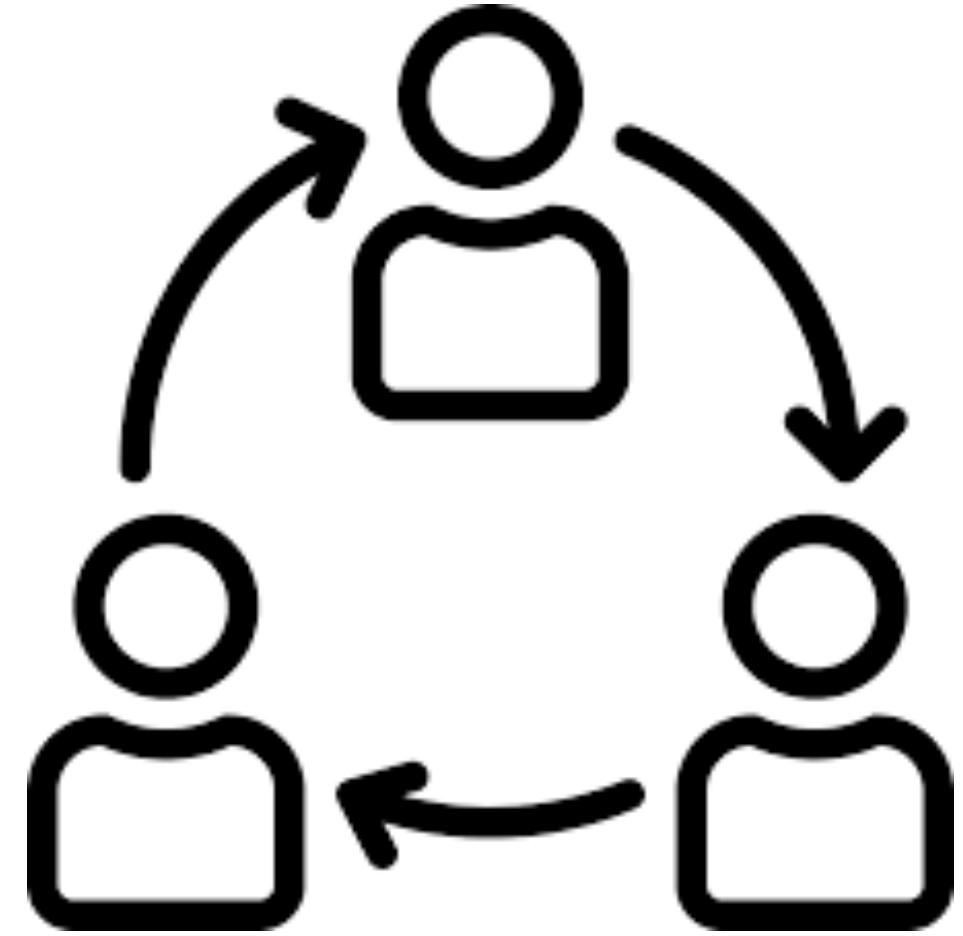
You can also always browse the current contents of the git repository using the [web interface](#).

</> About this site  
Patches, suggestions, and comments are welcome.

Git is a member of Software Freedom Conservancy



# Why version control with git and GitHub?



Collaboration



Returning to  
a safe state

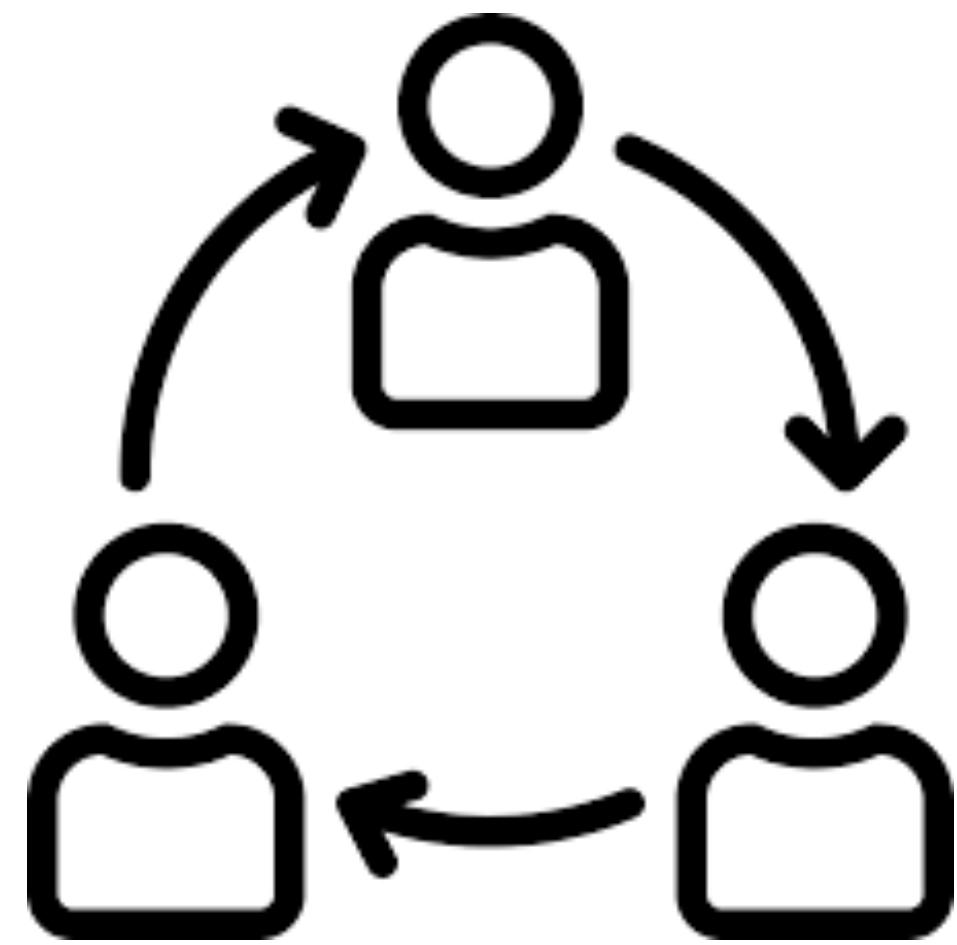


Exposure  
for your  
work

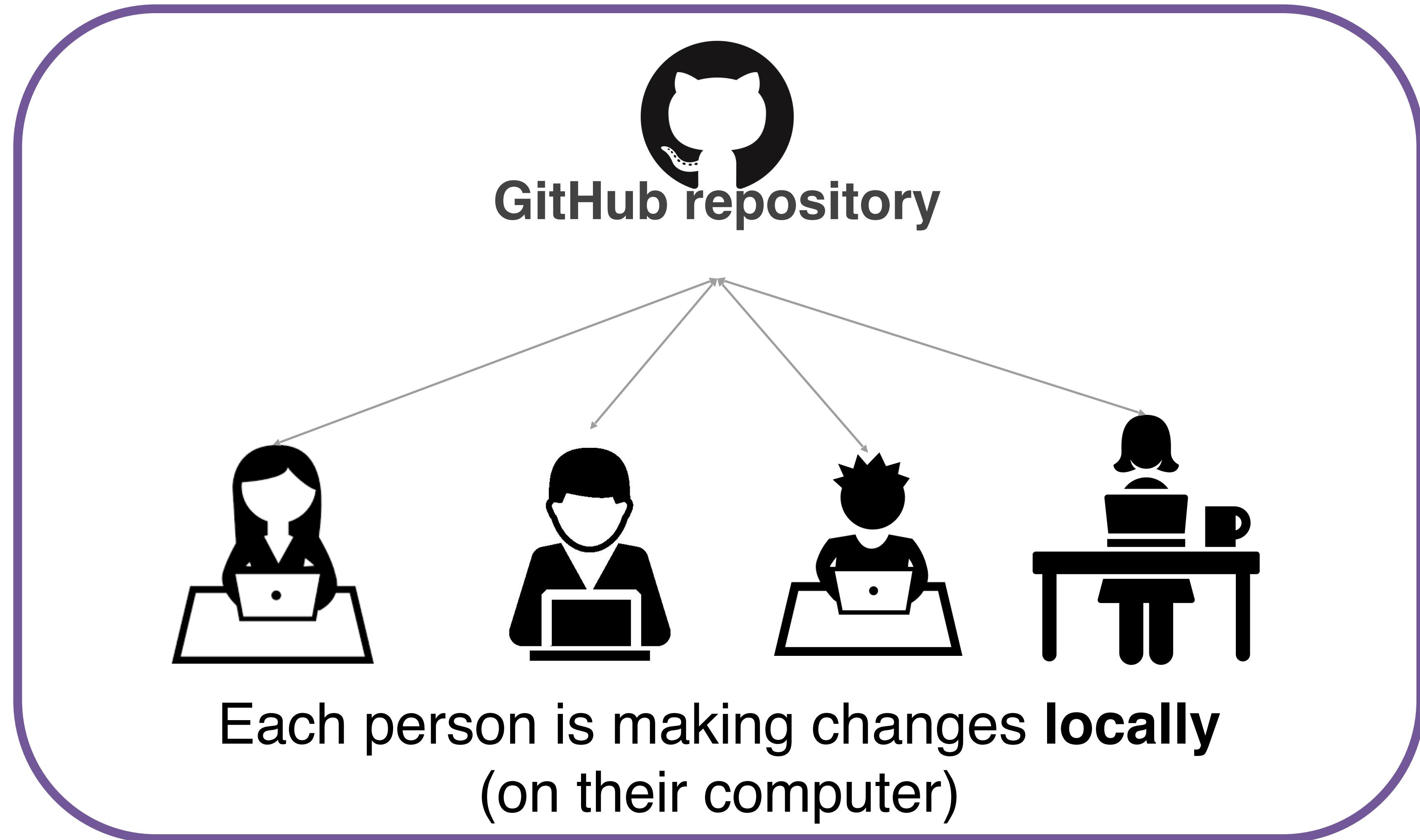


Tracking  
others' work

# Collaborate like you do with Google Docs



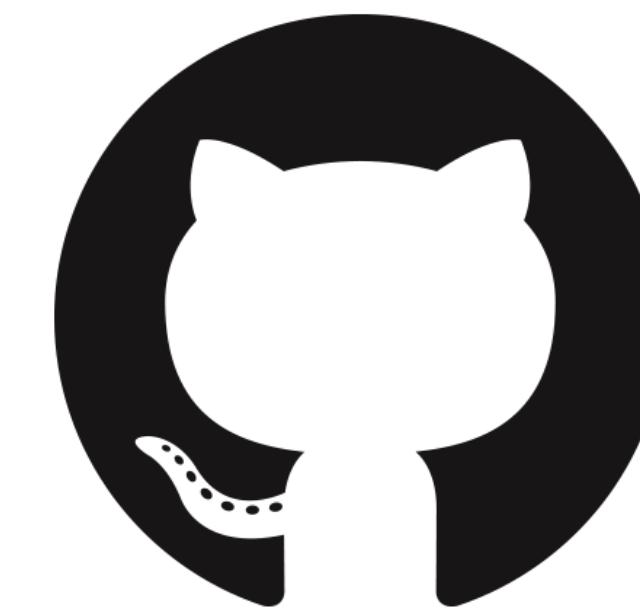
Collaboration



# Make changes locally, while knowing a stable copy exists



Returning to  
a safe state



You're free and safe to **try things out locally**.  
You'll only send changes to the repo when  
you're at a stable point

# Your repositories will be visible to others!



Exposure  
for your  
work



Your public GitHub repos  
are your coding social  
media

# And vice versa, you can search for the code you need

For instance, this might come in handy when thinking about class projects

<https://github.com/topics/datascience-projects>



Search or jump to...



Pull requests Issues Marketplace Explore

Explore

Topics

Trending

Collections

Events

GitHub Sponsors

#

# april-fools

 DuckMasterAI / rickroll-bot

 Star 5



 Code

 Issues

 Pull requests

A simple bot to rickroll your friends on Discord!

discord-bot

discord-py

april-fools

rickroll

never-gonna-give-you-up

never-gonna-let-you-down

rick-astley

discord-py-bot

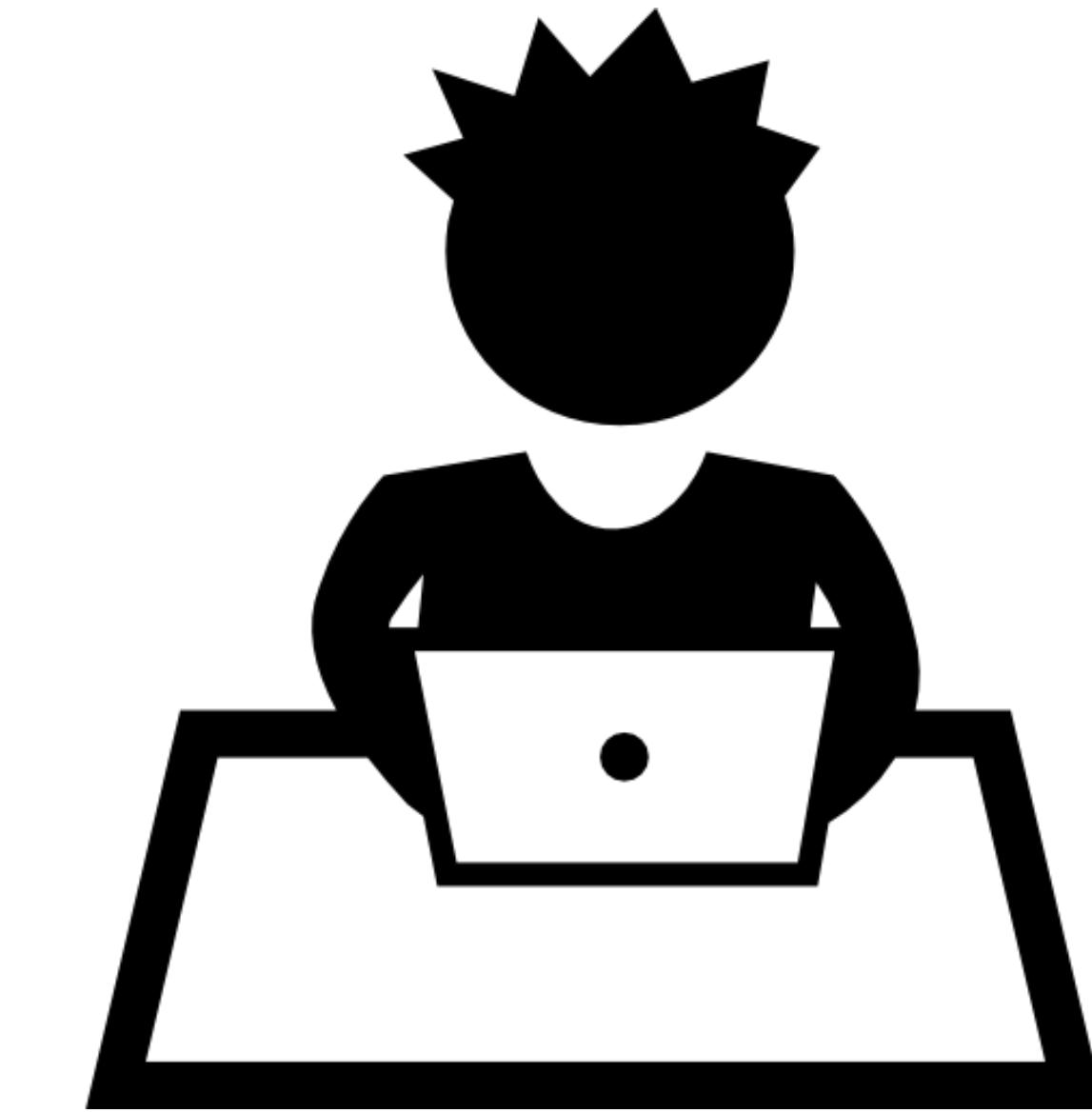
Updated 2 hours ago

Python

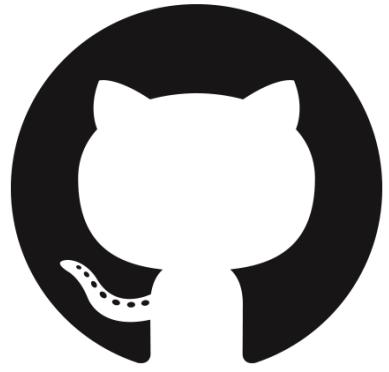
# Keep up with others' work easily



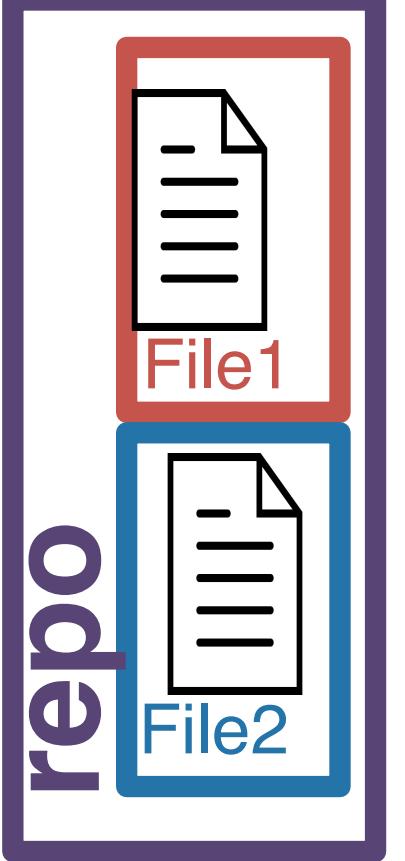
Tracking  
others' work



As a social platform, you  
can see others' work too!

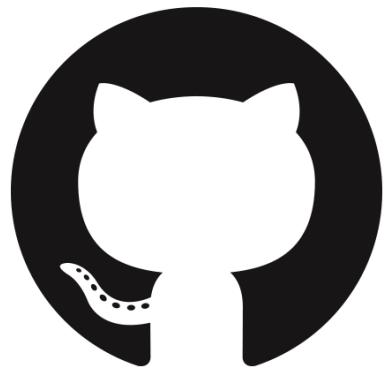


repo

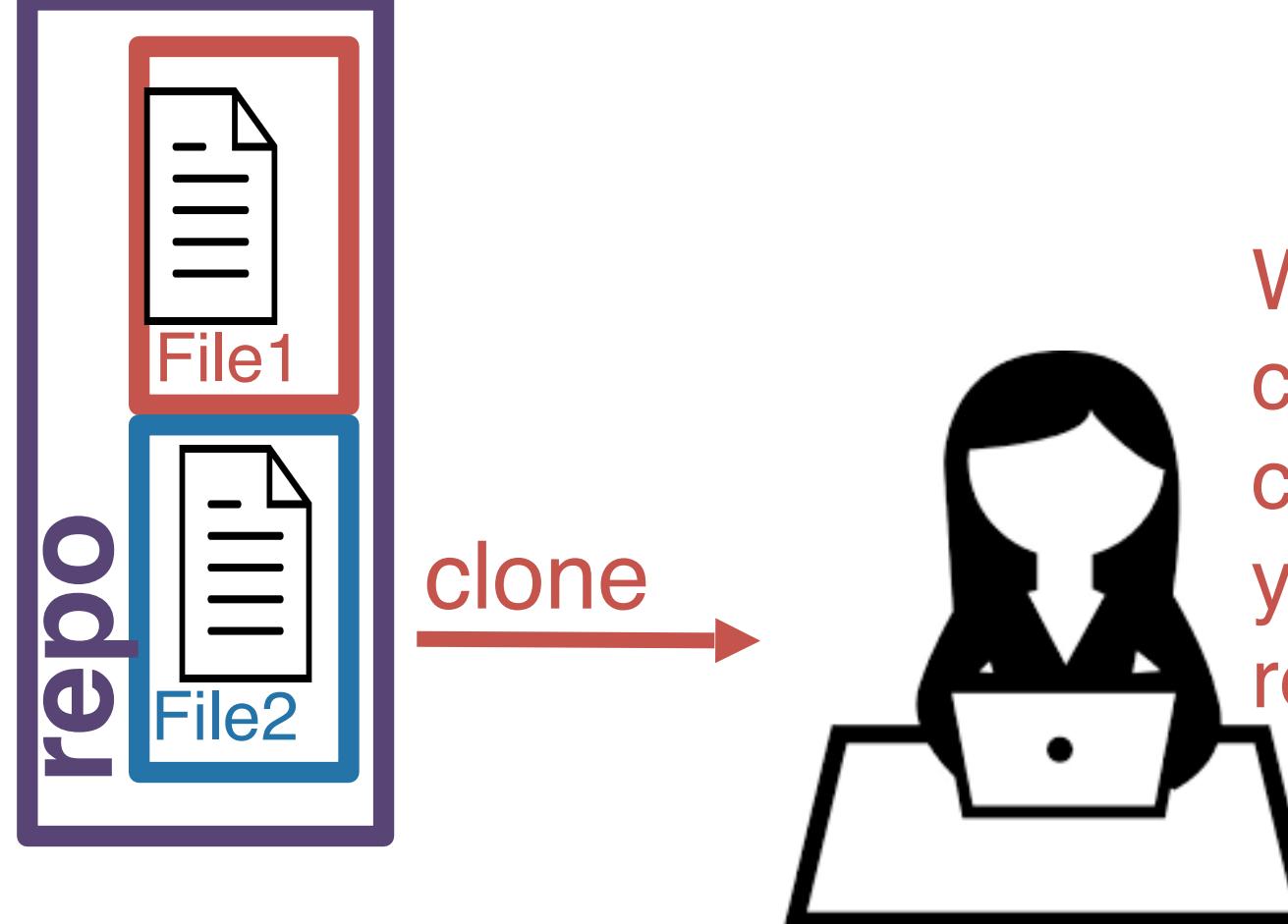


A **GitHub repo** contains all the files and folders for your project.

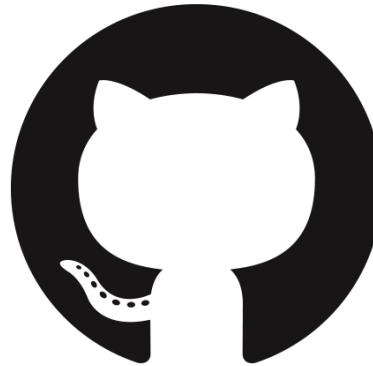
GitHub is a **remote host**. The files are geographically distant from any files on your computer.



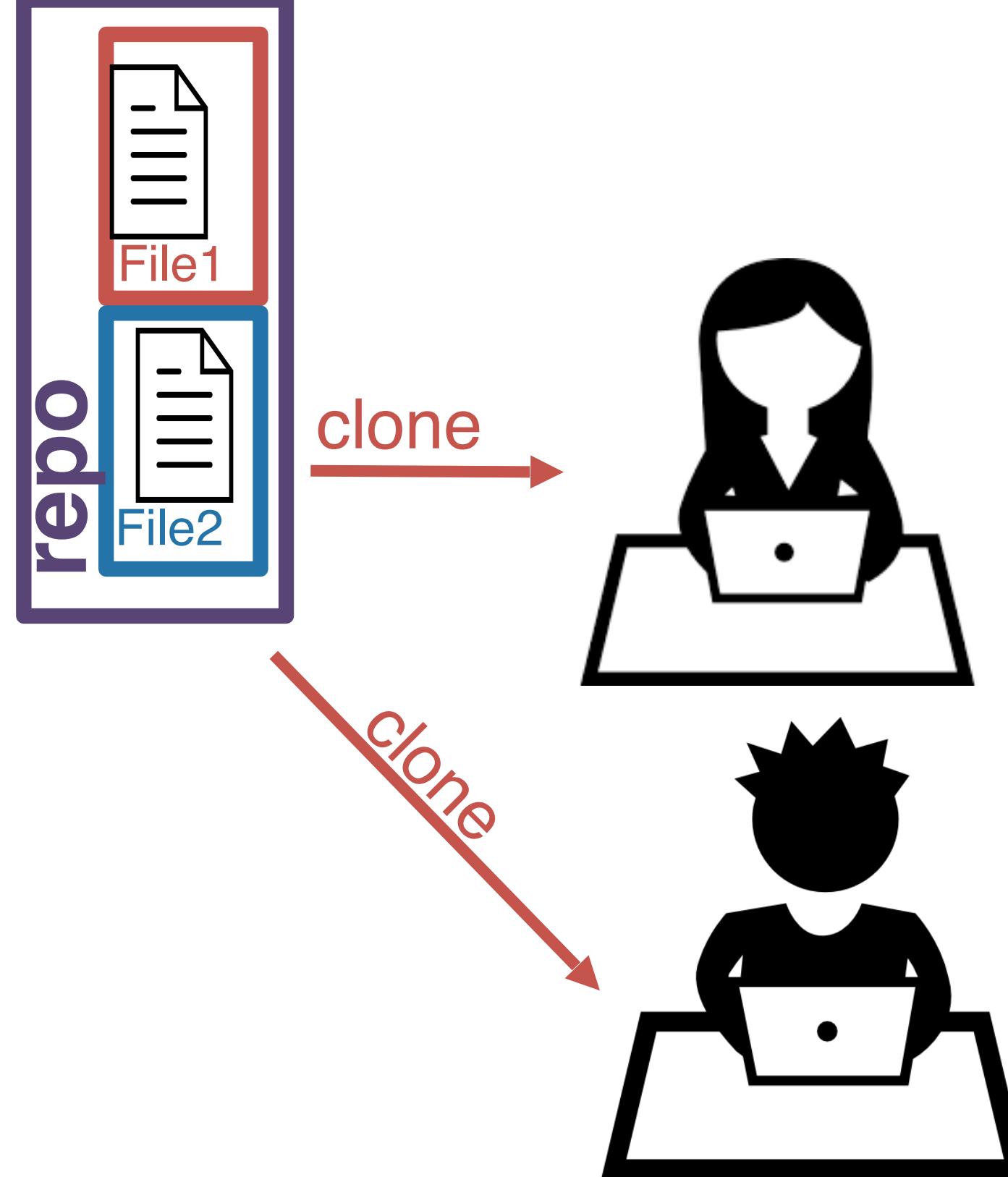
repo



When you first make a copy onto your local computer (read: laptop), you **clone** the repository.



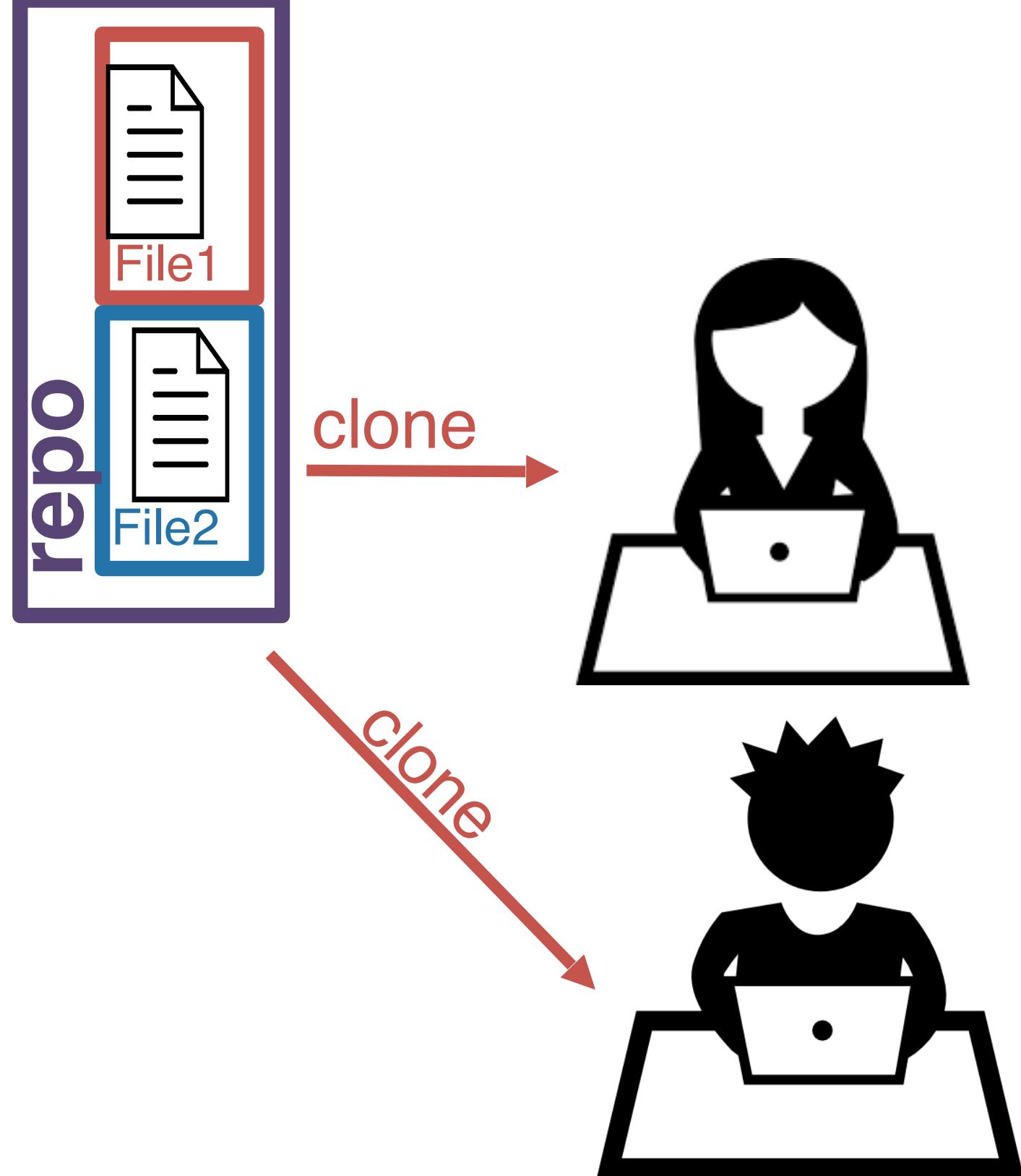
repo



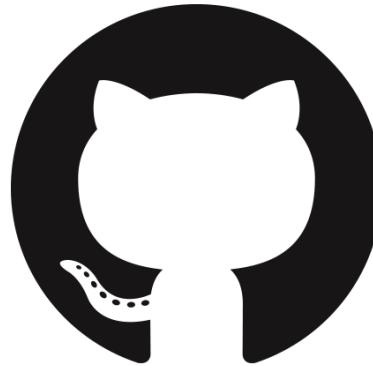
If someone else on your project cloned the repo at the same time, you would have identical copies of the project on each of your computers.



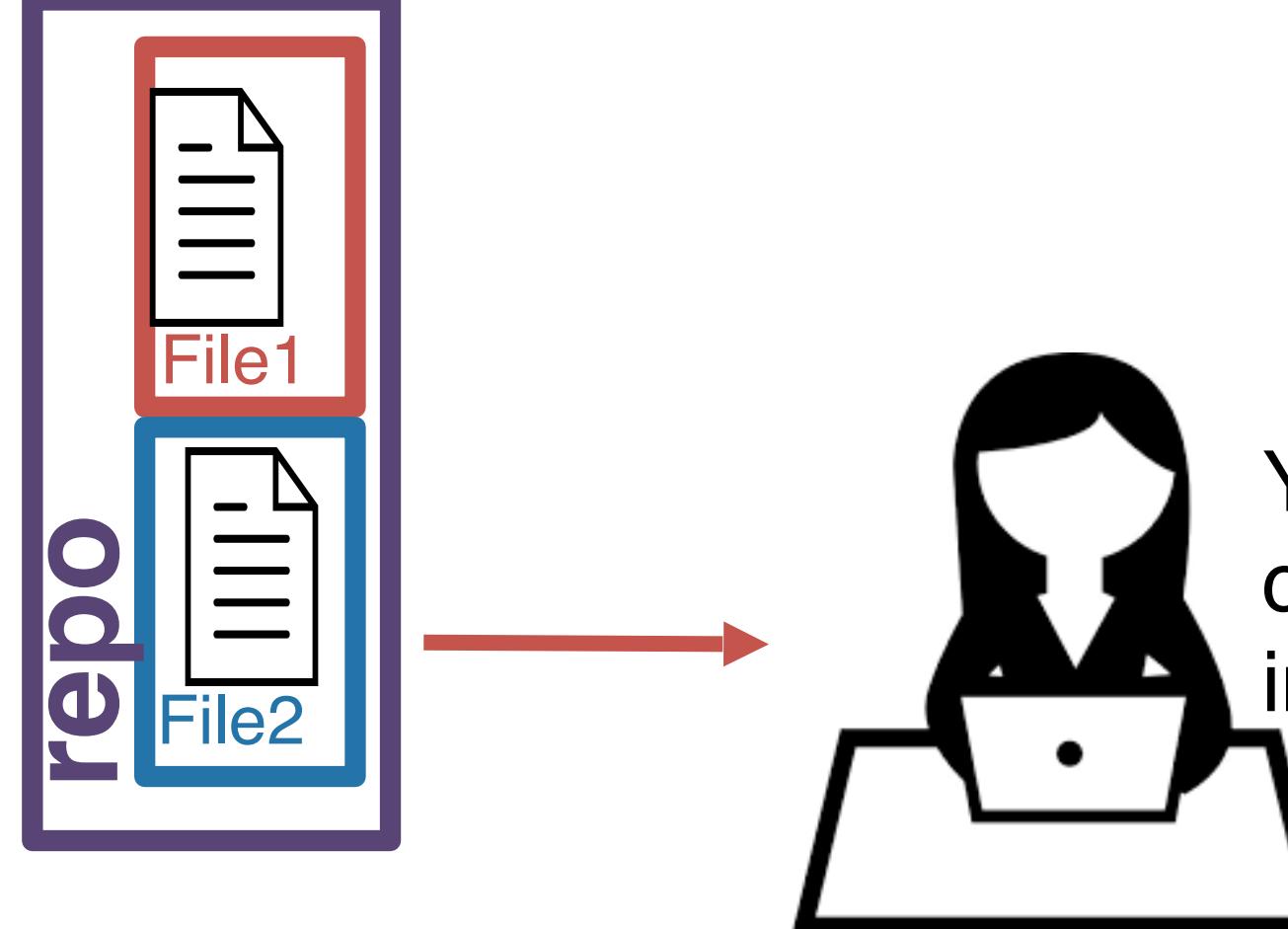
repo



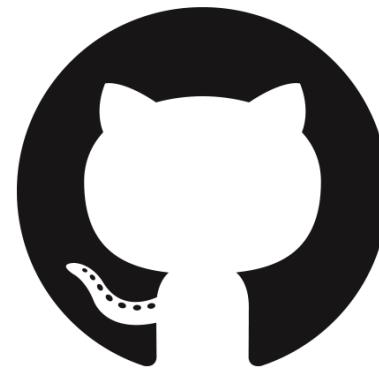
Yay! Everyone can  
work on the project!



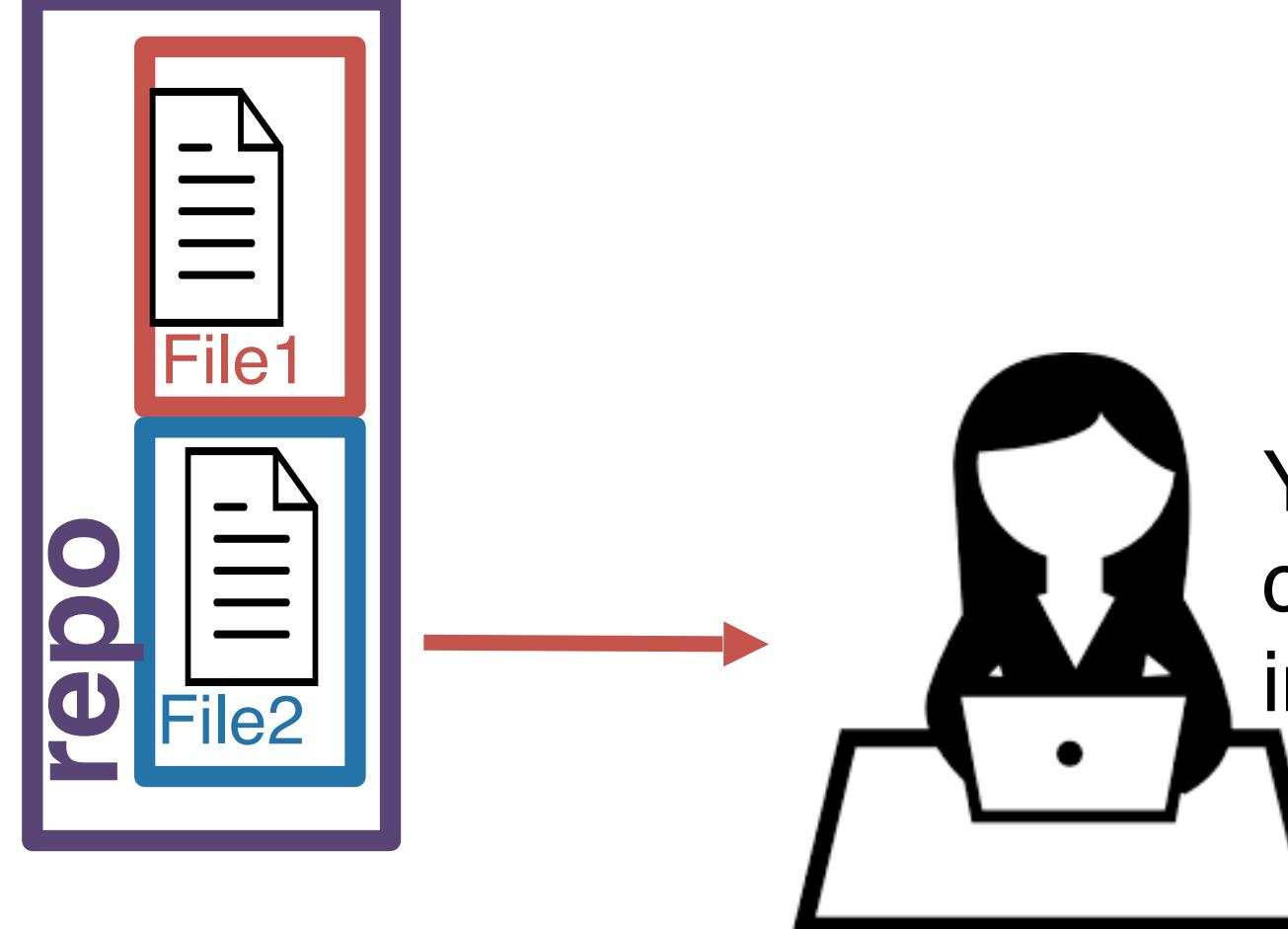
repo



You decide you want to  
change some of the text  
in the project.



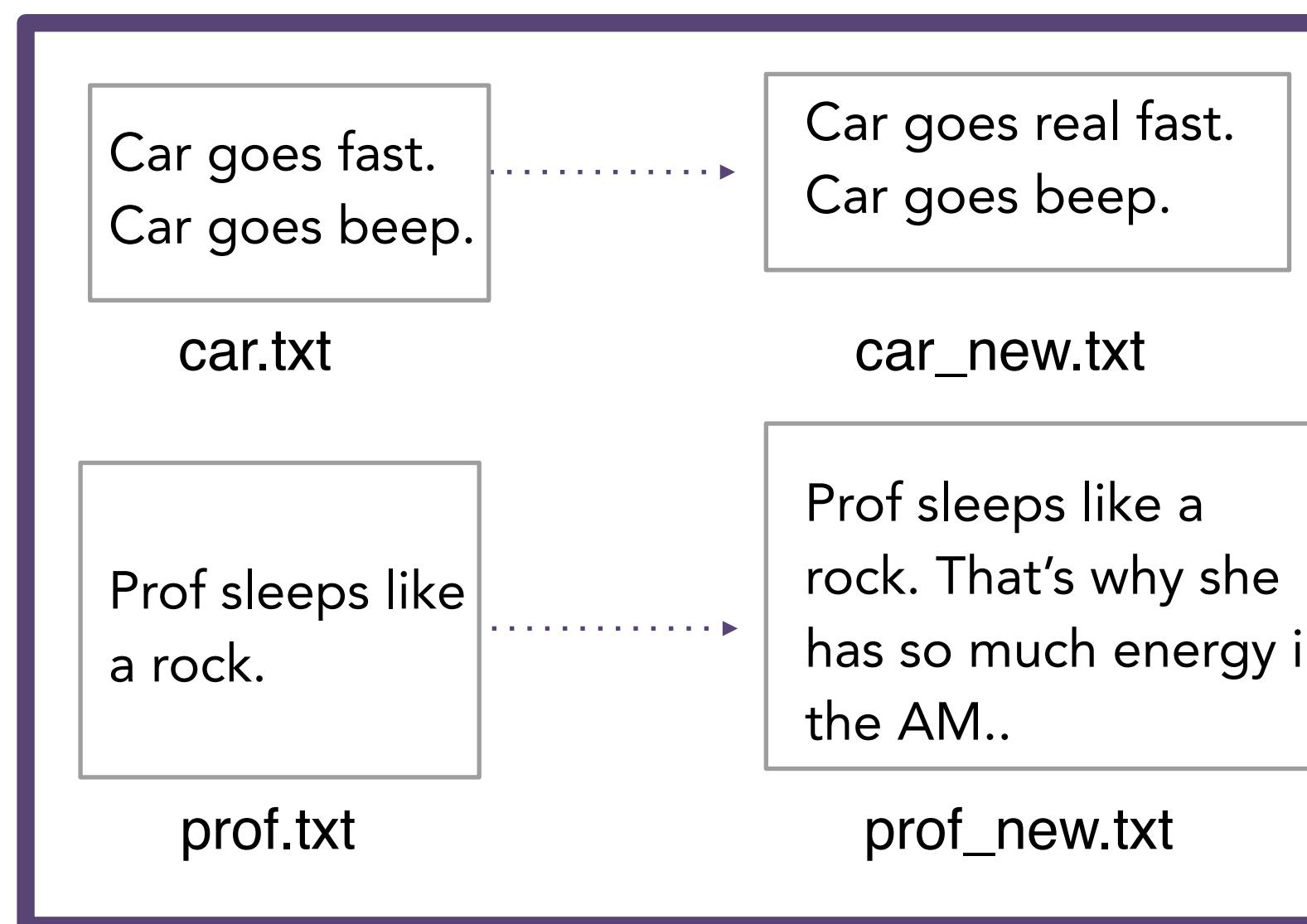
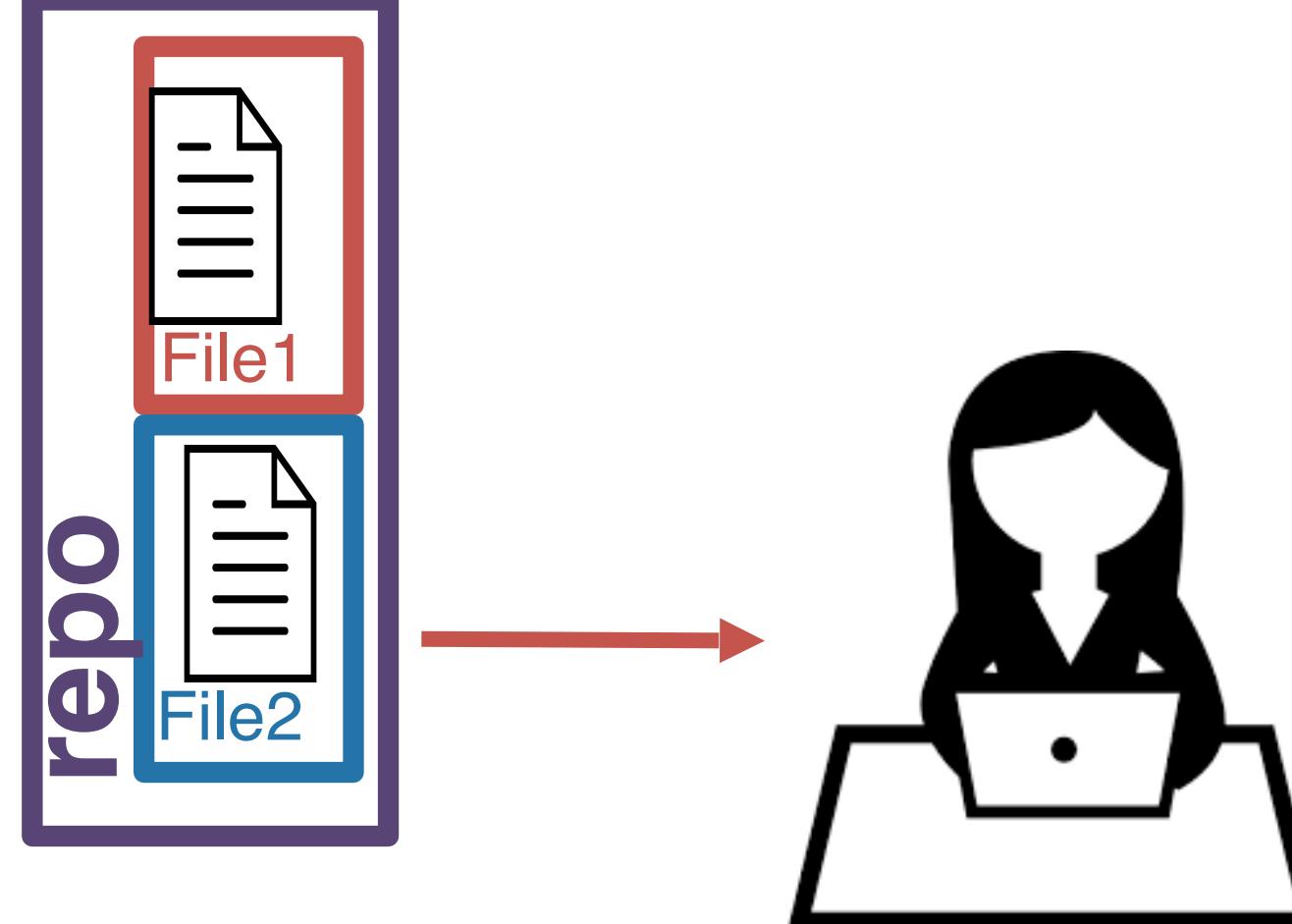
repo



You decide you want to  
change some of the text  
in the project.



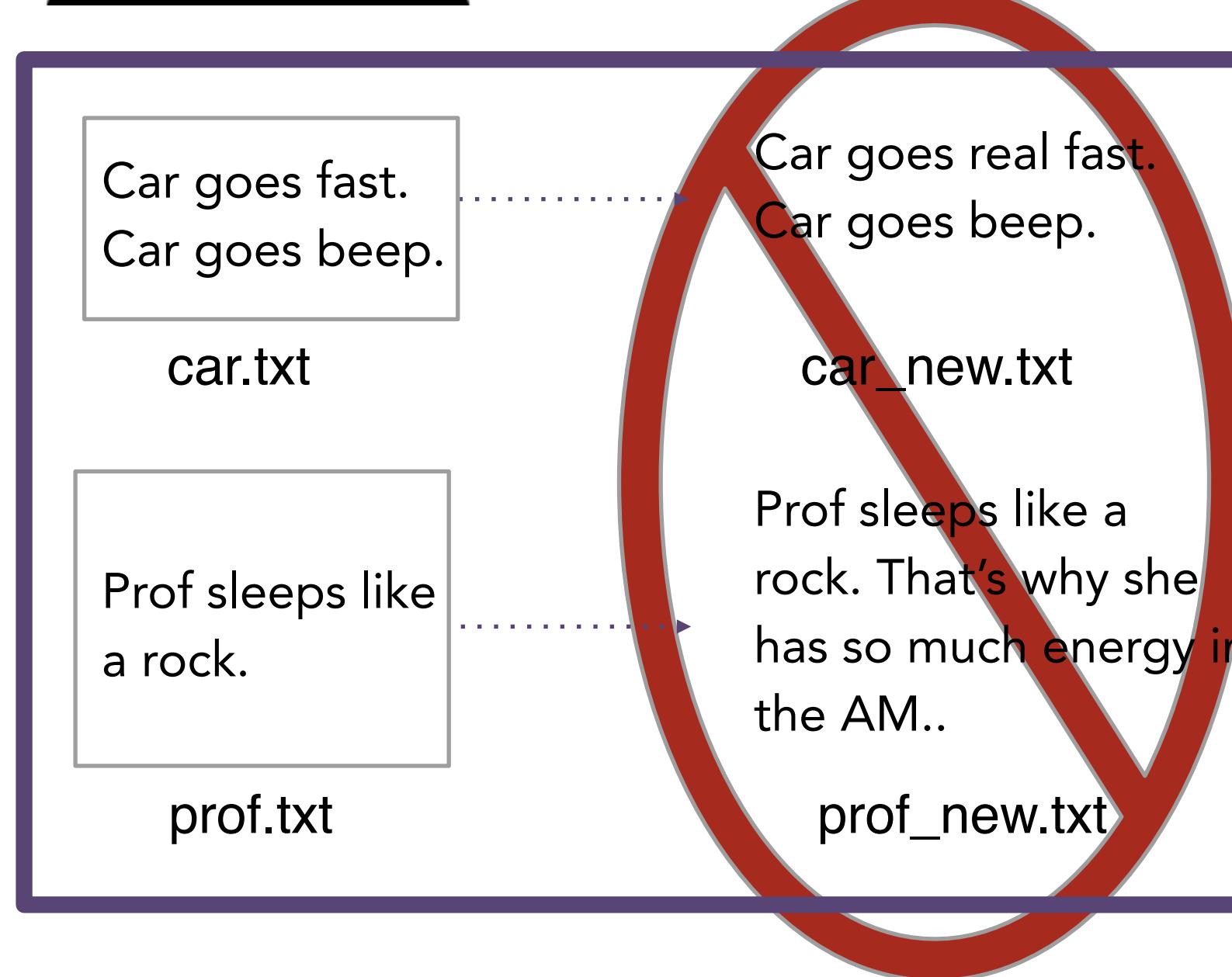
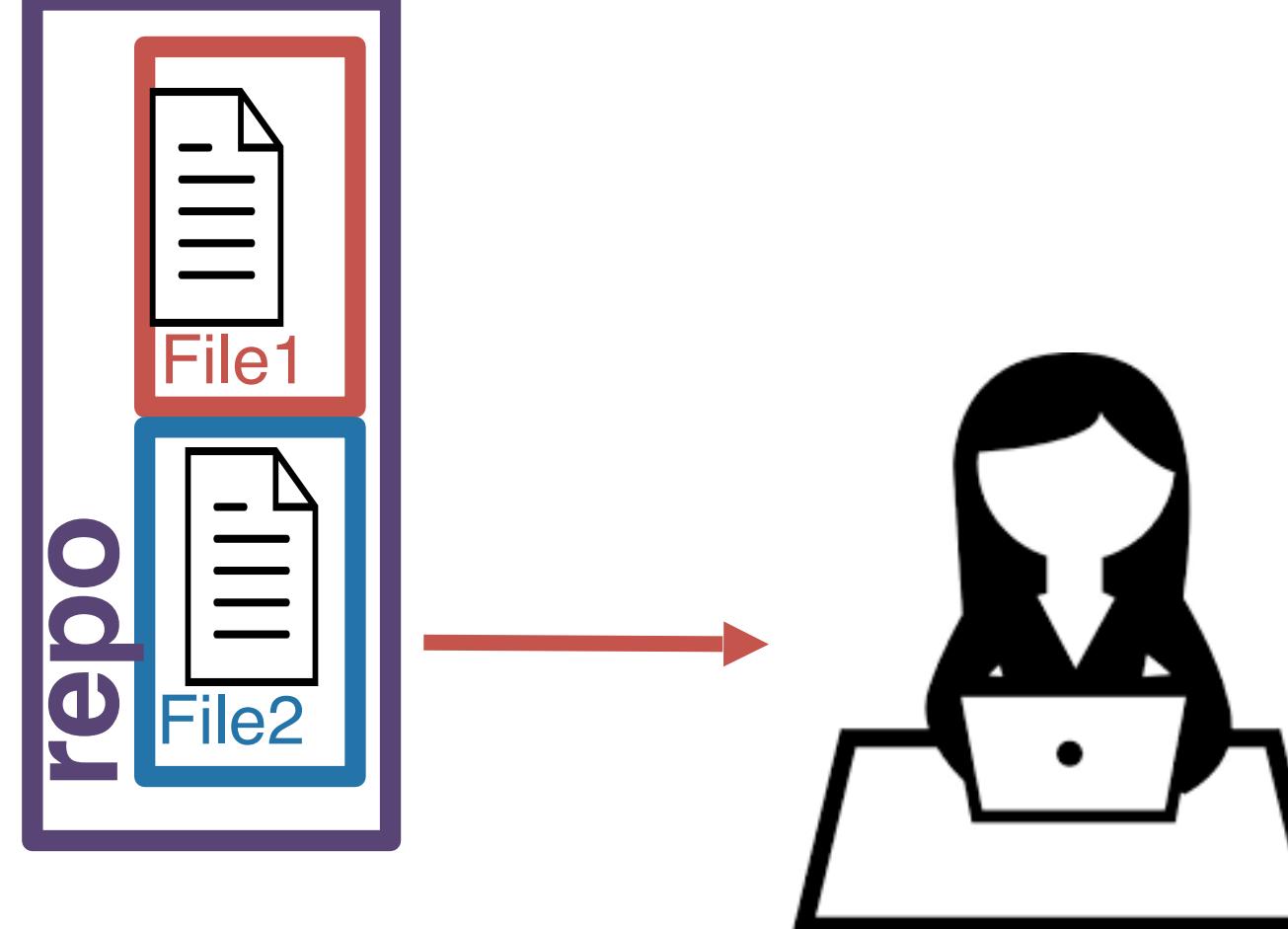
repo



without git...you'd  
likely rename  
these files....



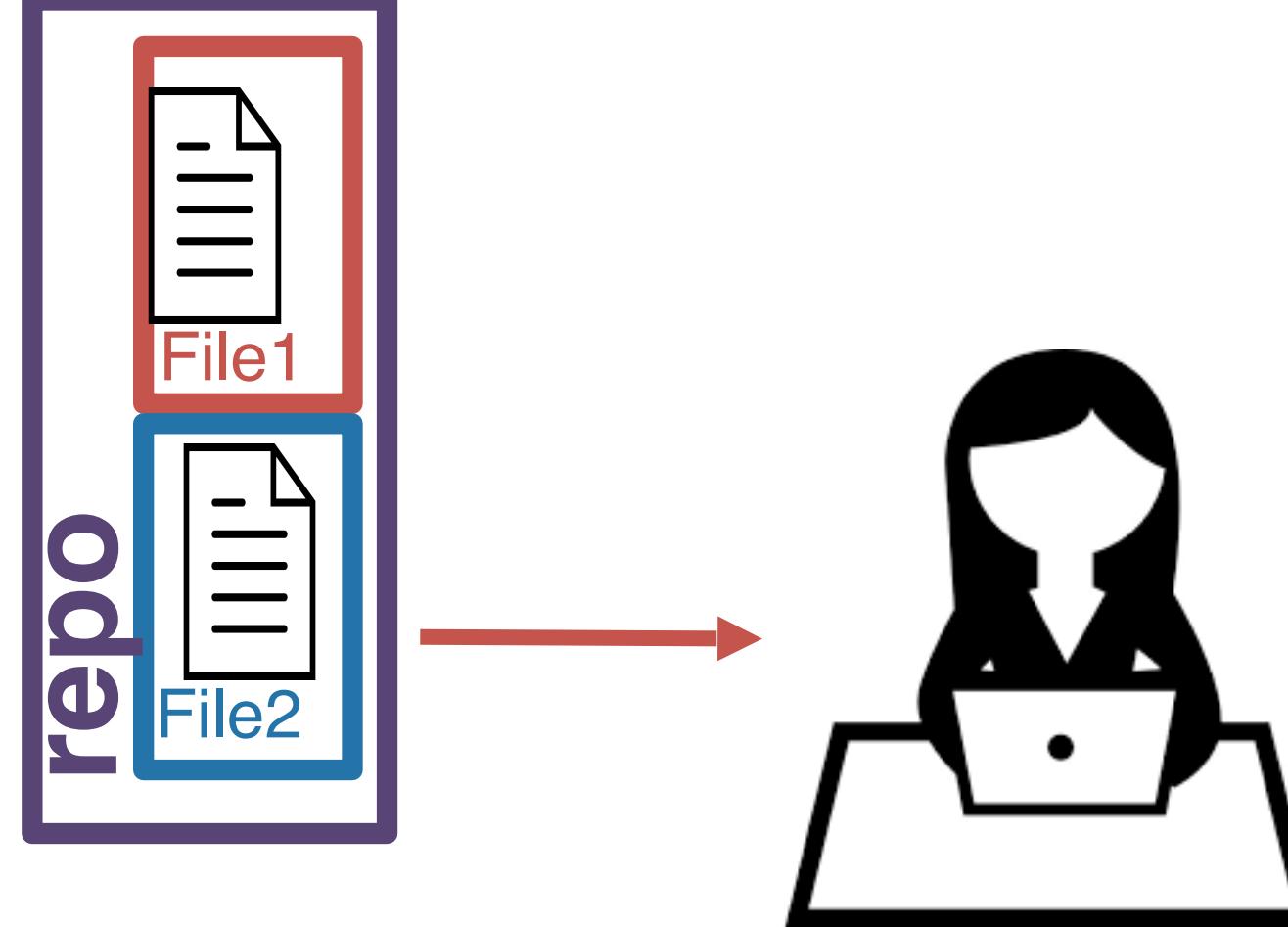
repo



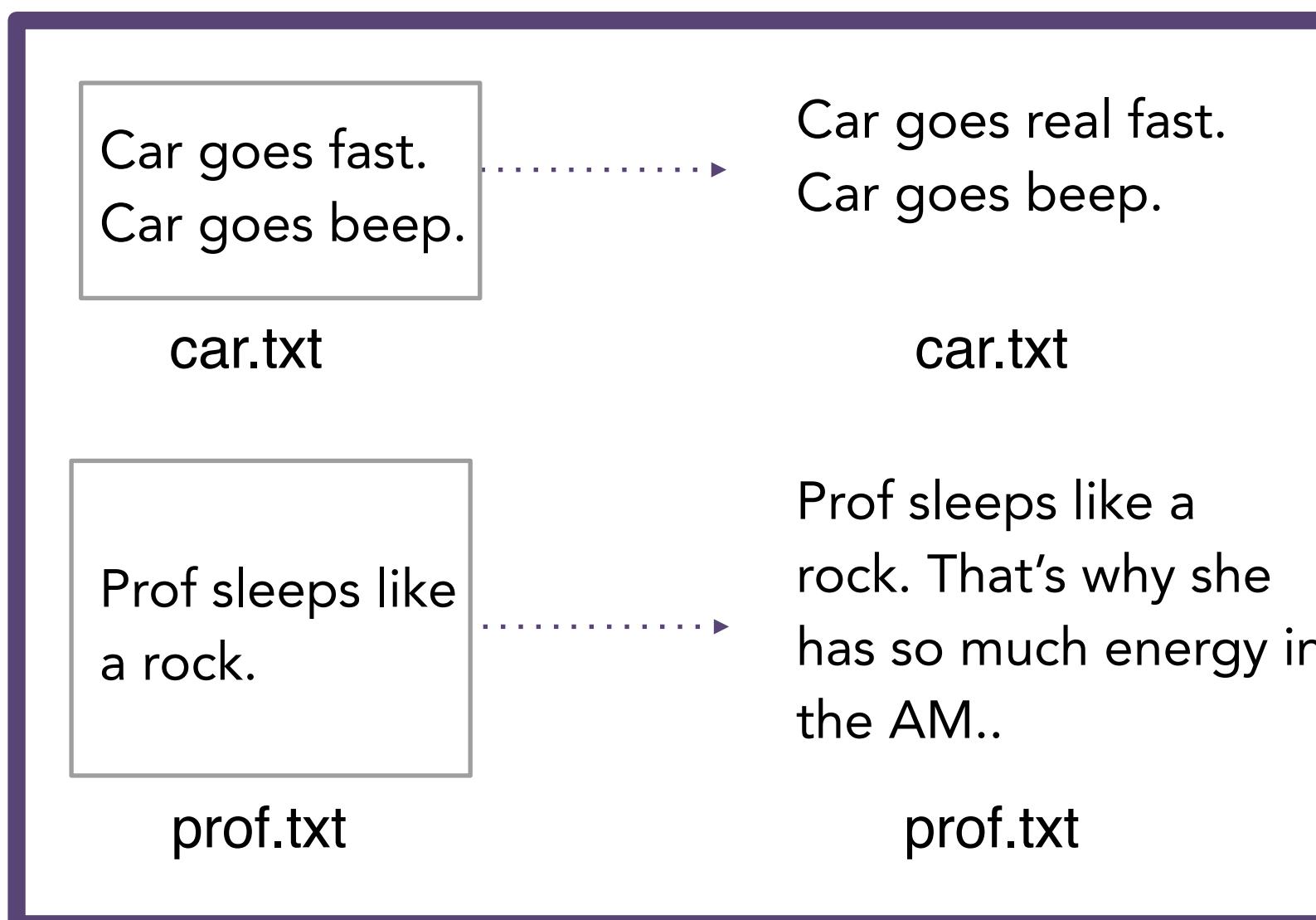
Thank  
goodness those  
days are over!



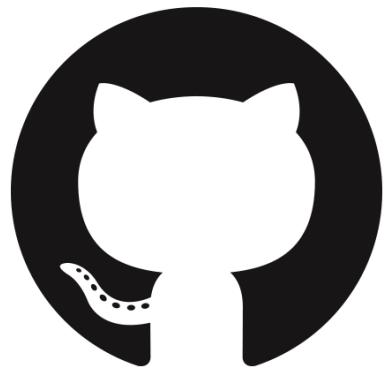
repo



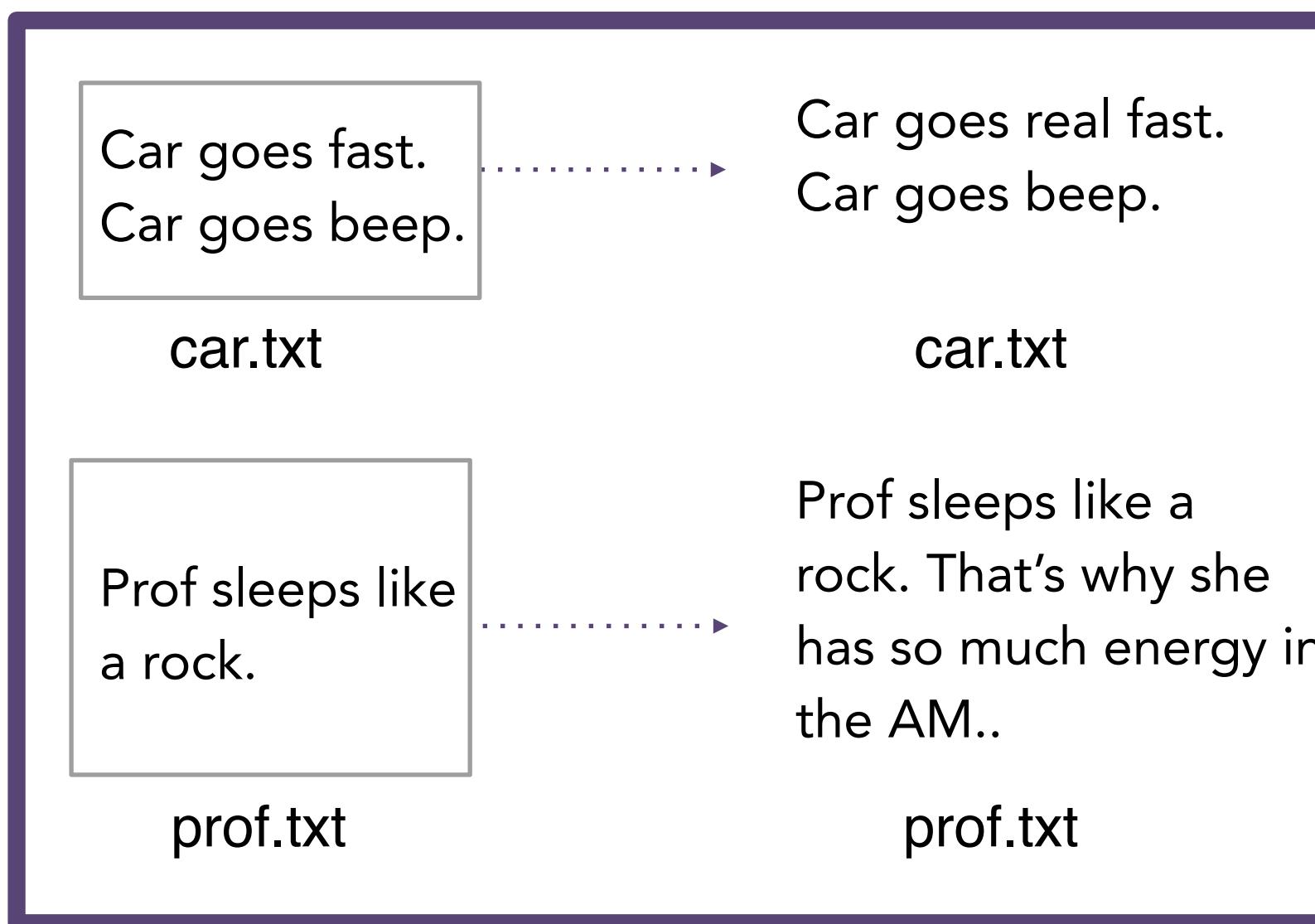
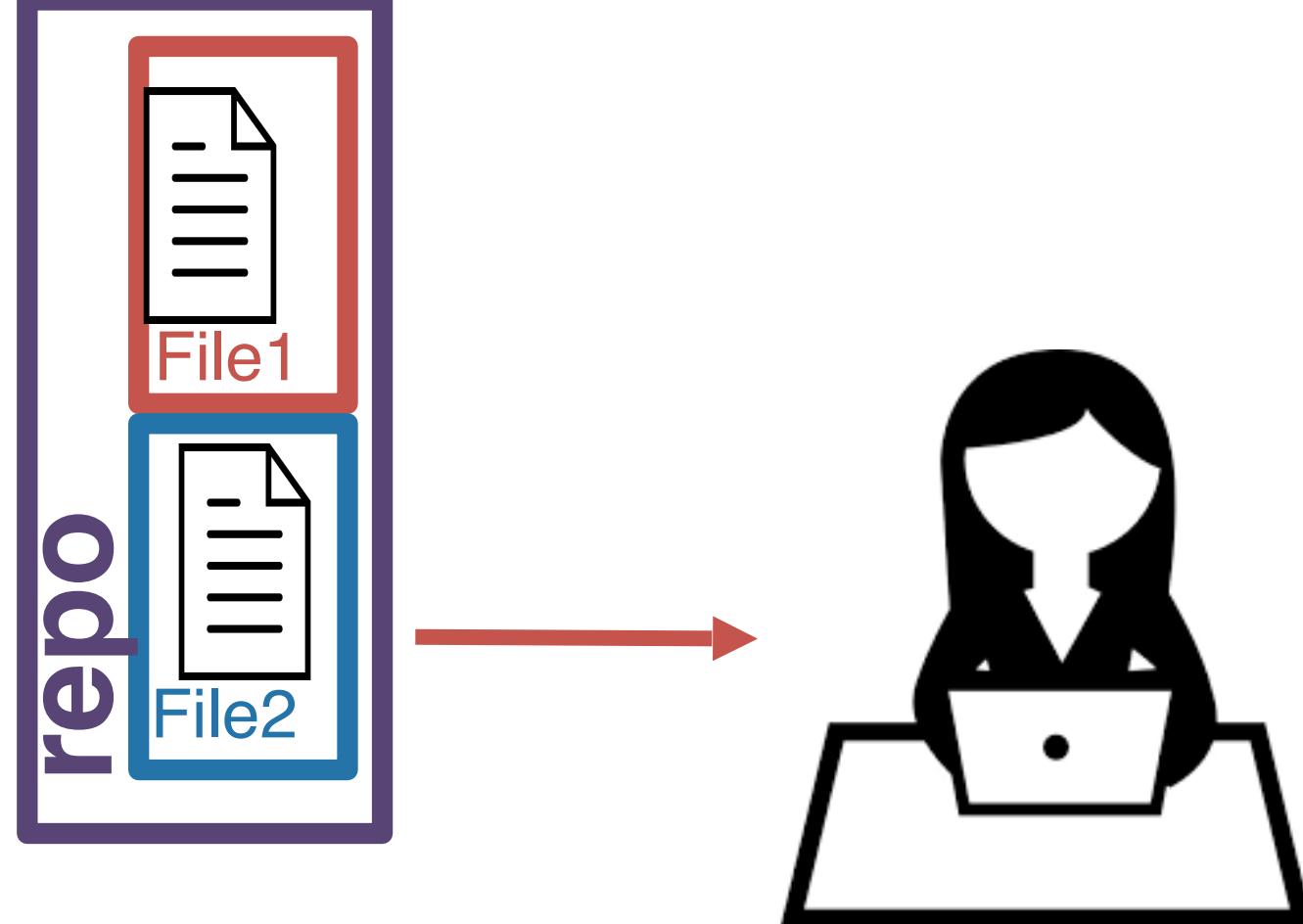
<code>git add file</code>	stages specified file (or folder)
<code>git add .</code>	stages new and modified files
<code>git add -u</code>	stages modified and deleted files
<code>git add -A</code>	stages new, modified, and deleted files
<code>git add *.csv</code>	stages any files with .csv extension
<code>git add *</code>	use with caution: stages everything



Instead, you tell git which files you'd like to keep track of using **add**. This process is called *staging*.



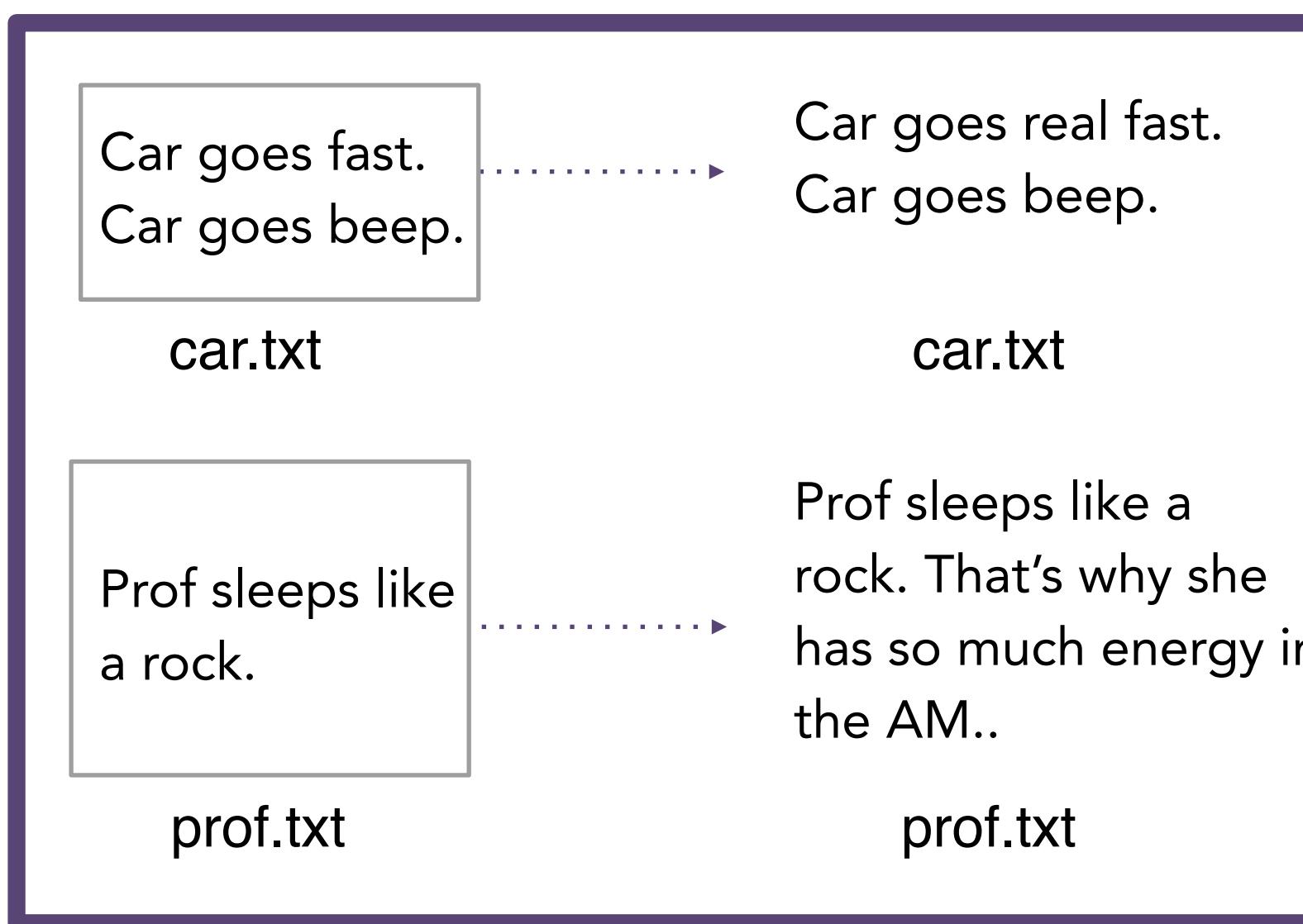
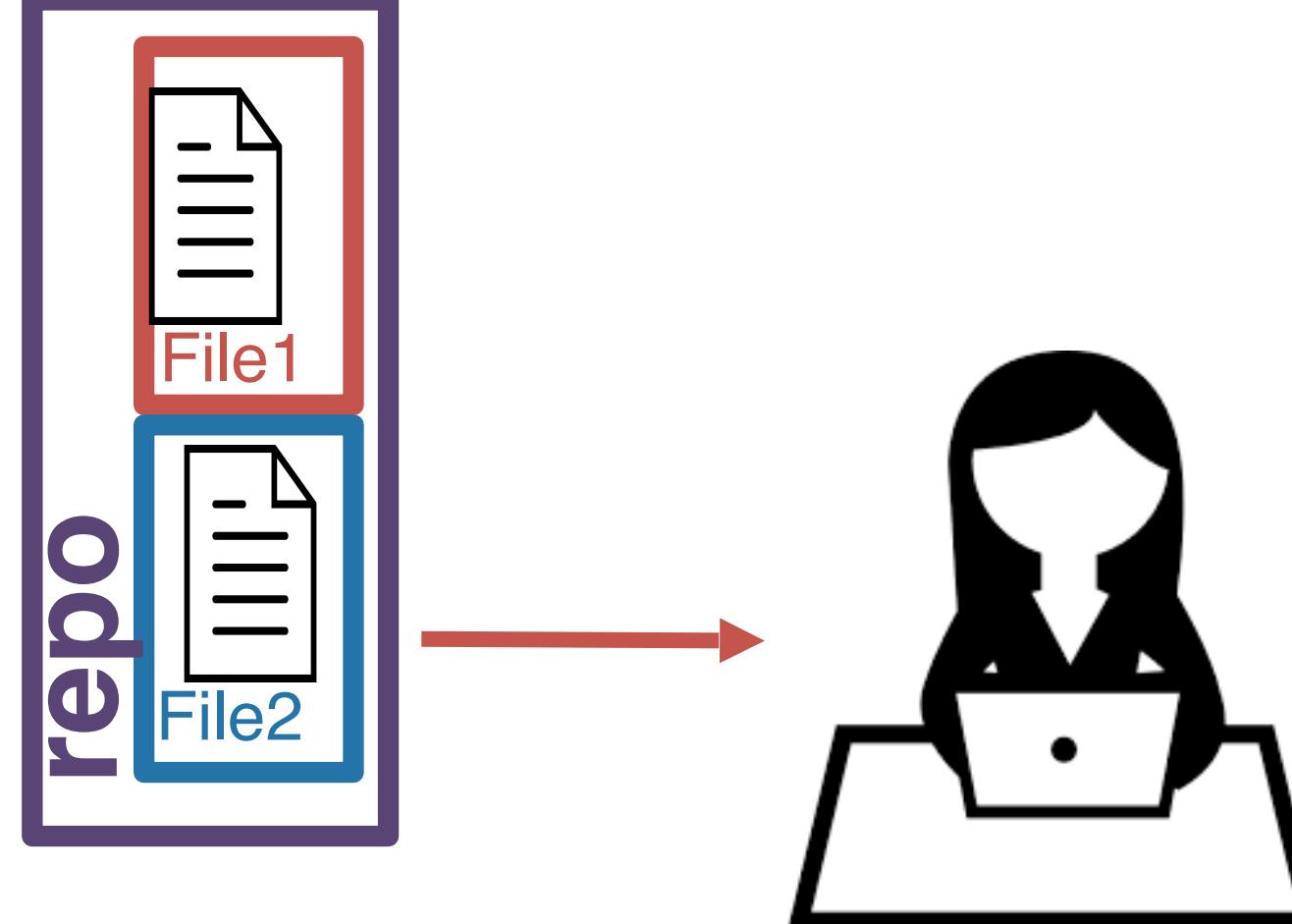
repo



Then, you create a snapshot of your files at this point. This snapshot is called a **commit**.



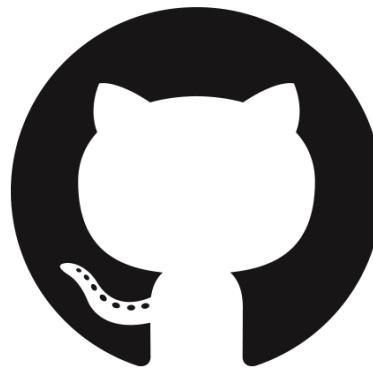
repo



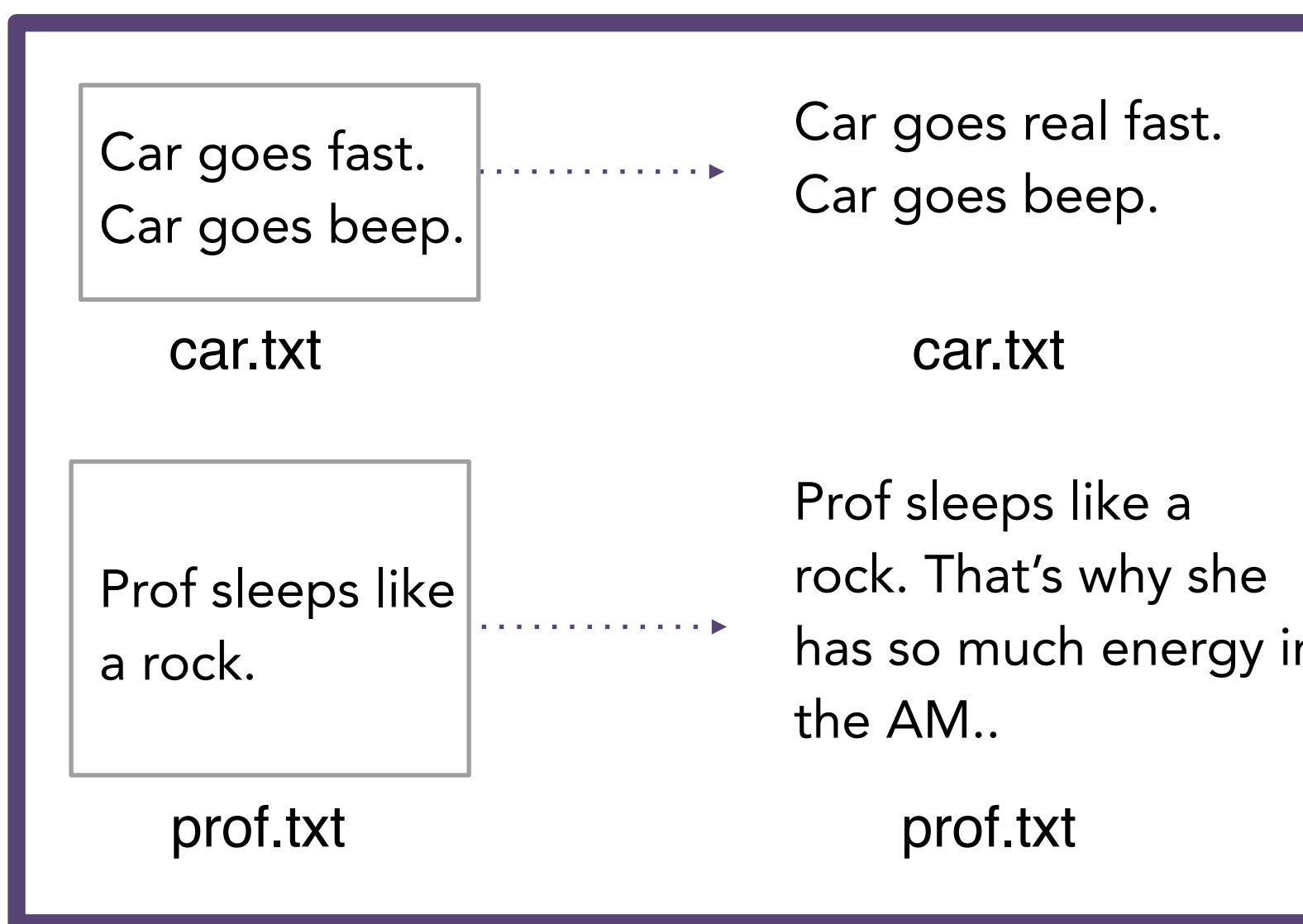
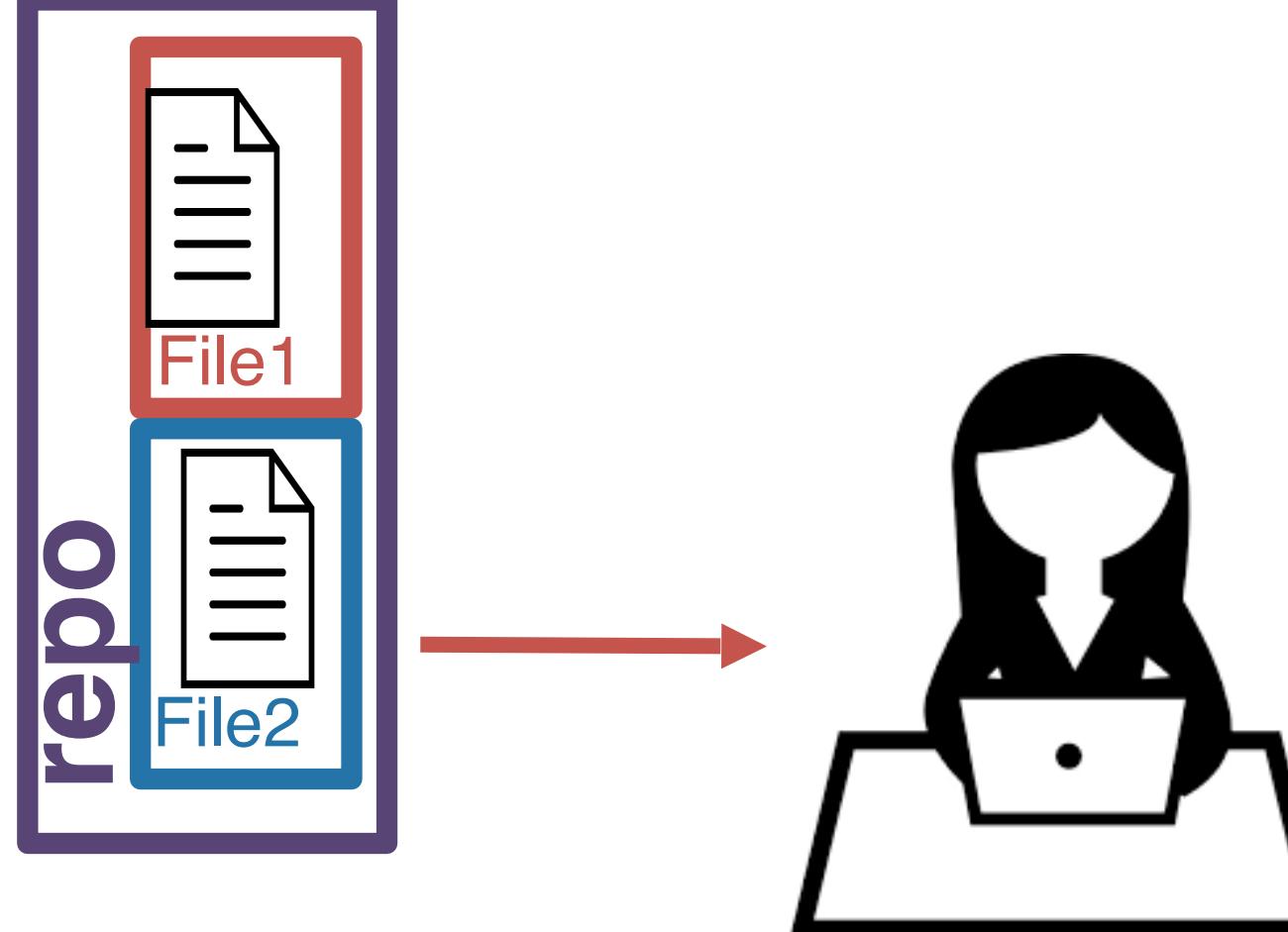
Then, you create a snapshot of your files at this point. This snapshot is called a **commit**.



A **commit** tracks  
who, what, and  
when



repo



A **commit** tracks  
who, what, and  
when

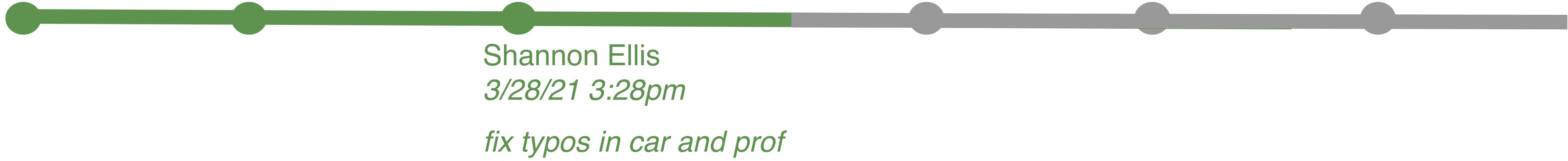
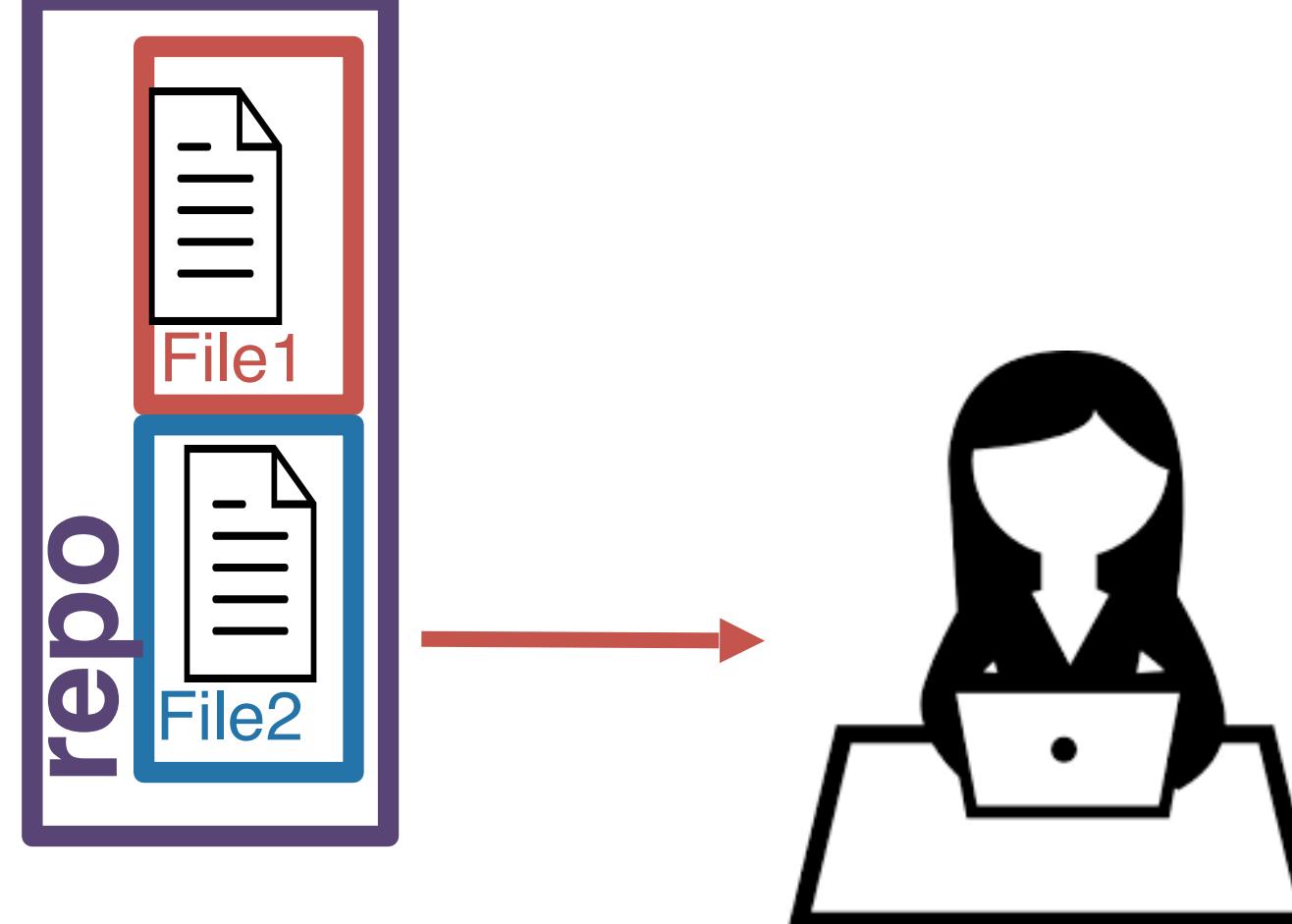
You can make commits more informative by adding a **commit message**.

Example: `git commit -m 'fix typos in car and prof'`

Then, you create a snapshot of your files at this point. This snapshot is called a **commit**.

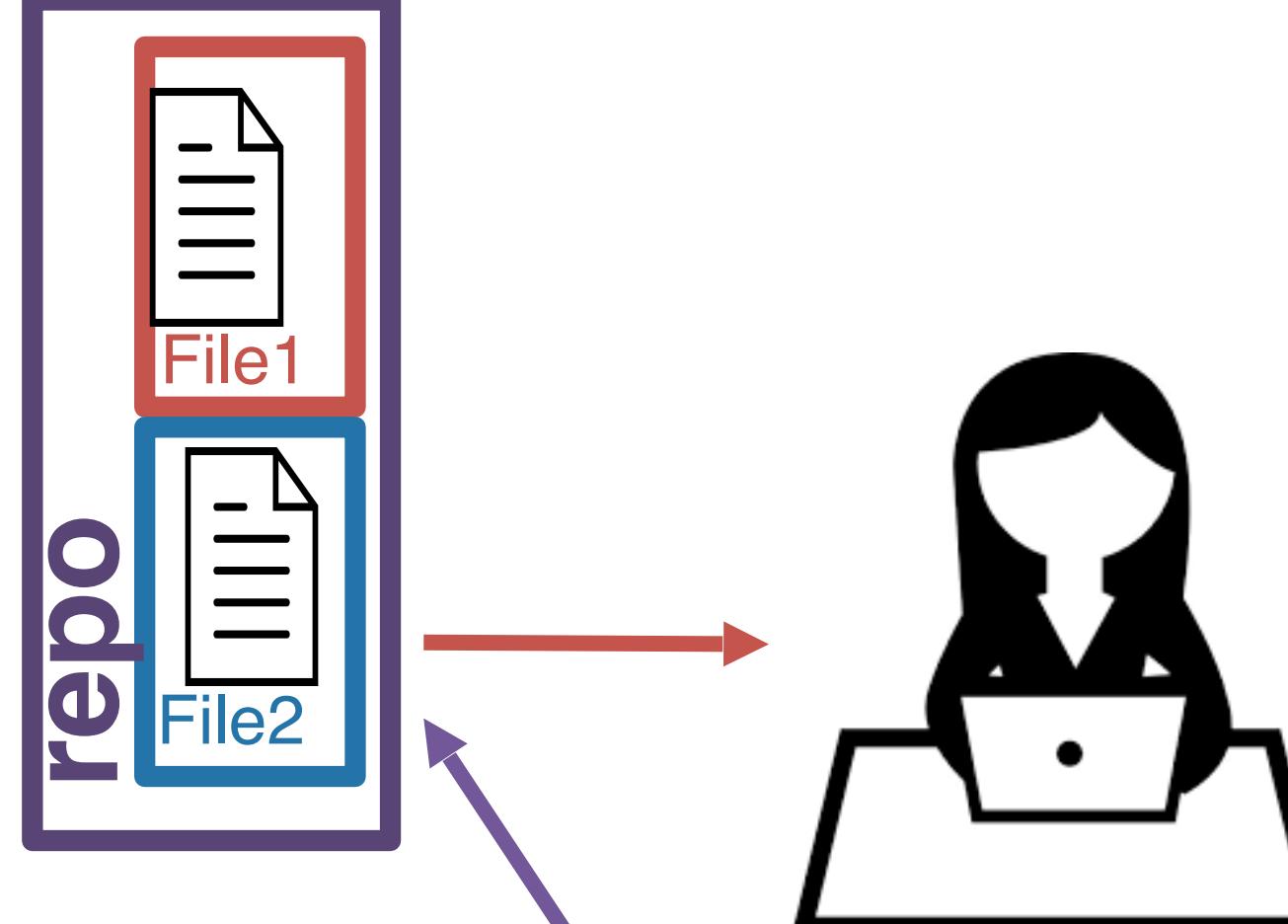


repo





repo

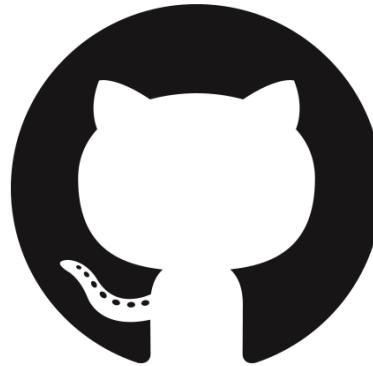


Remember, you're not the only one working on this project though! You want your teammates to have access to these changes! You **push** these changes back to the remote.

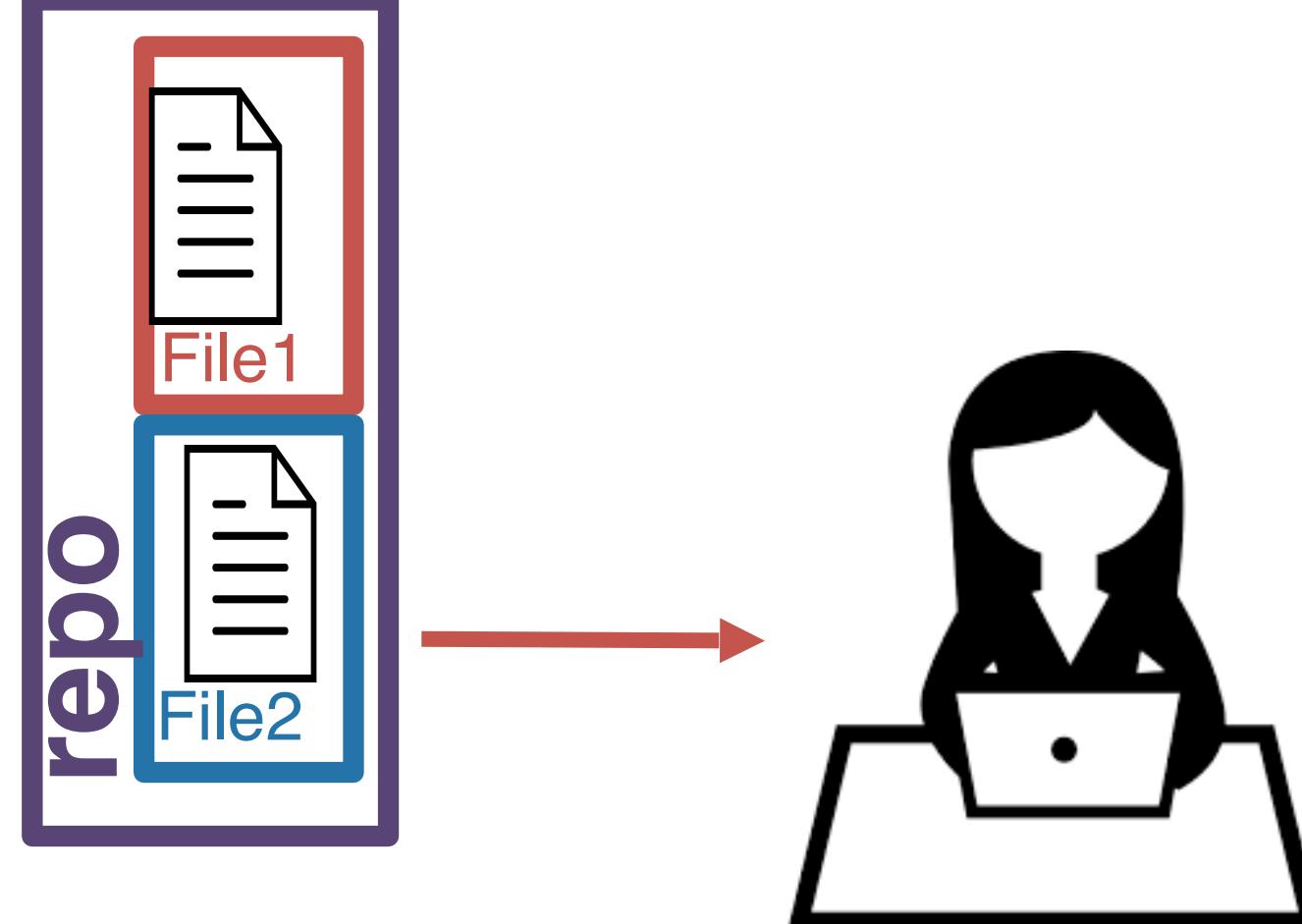


Shannon Ellis  
3/28/21 3:28pm

*fix typos in car and prof*

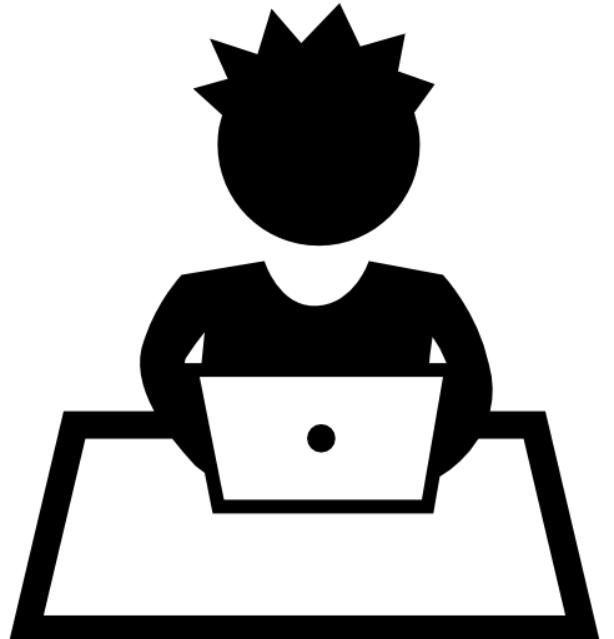


repo



Shannon Ellis  
3/28/21 3:28pm

*fix typos in car and prof*



Your teammate is still  
working with the (out-  
of-date) copy he  
cloned earlier!

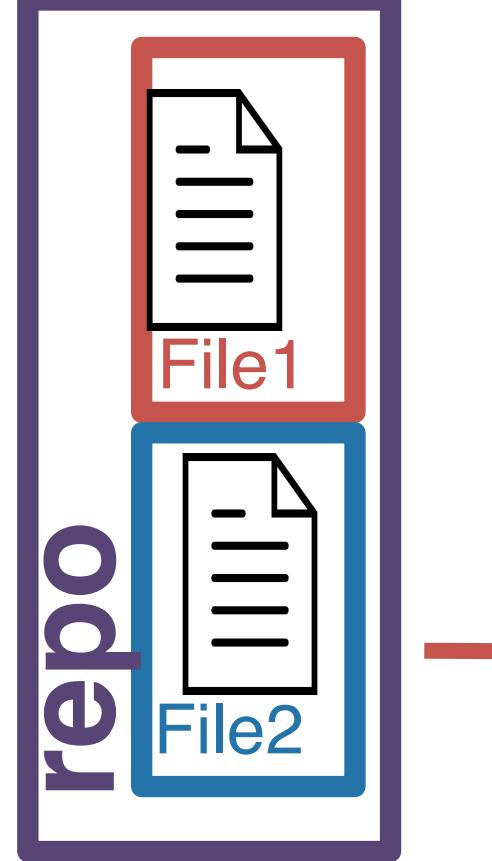


Shannon Ellis  
3/28/21 3:28pm

*fix typos in car and prof*



repo



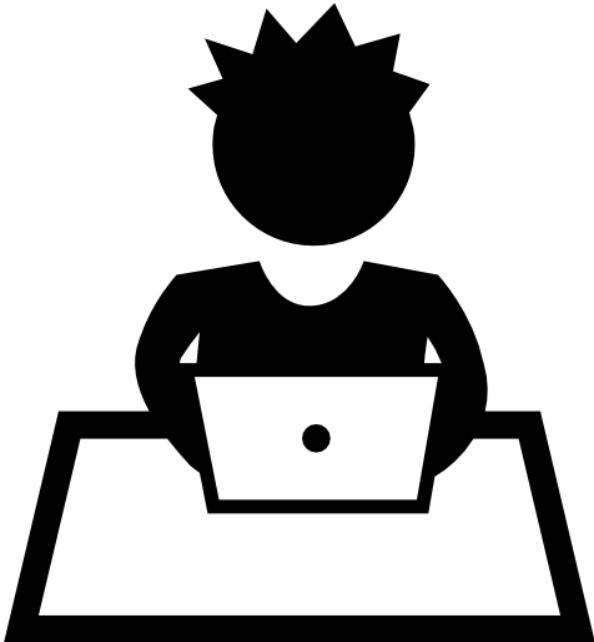
To catch up, your teammate will have to **pull** the changes from GitHub (remote)



Shannon Ellis

3/28/21 3:28pm

*fix typos in car and prof*



Your teammate is still working with the (out-of-date) copy he cloned earlier!



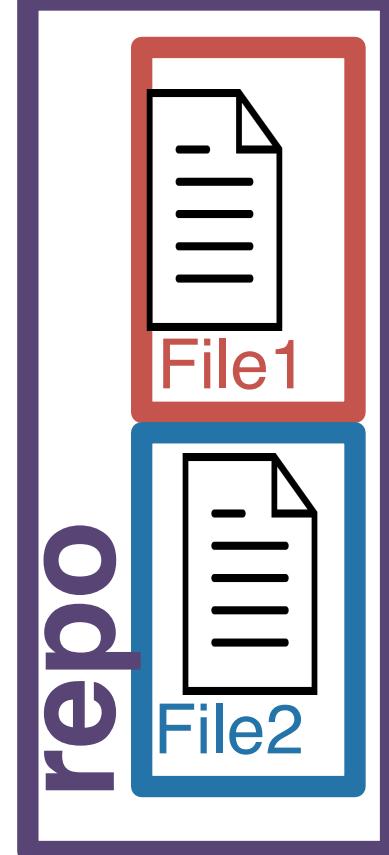
Shannon Ellis

3/28/21 3:28pm

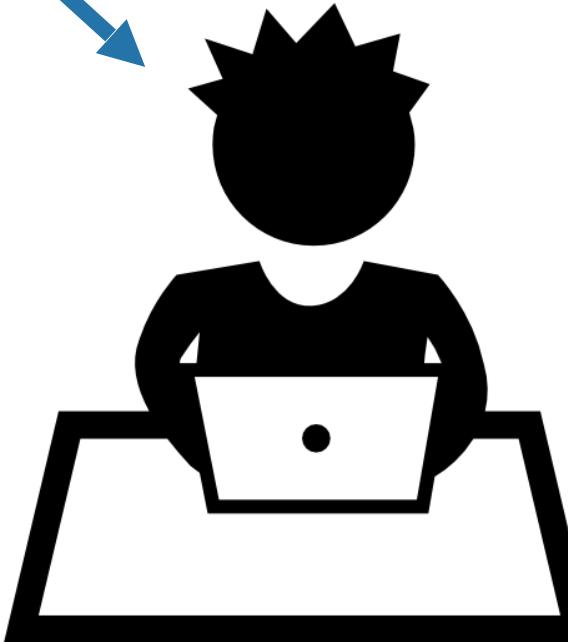
*fix typos in car and prof*



repo



*pull*

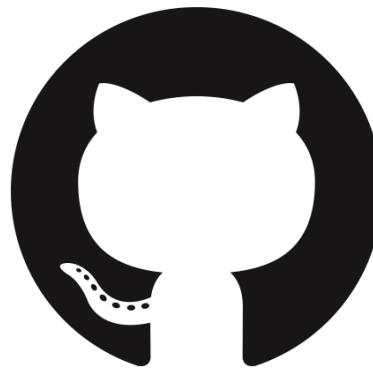


Your teammate pulls  
from remote and is  
now up-to-date!

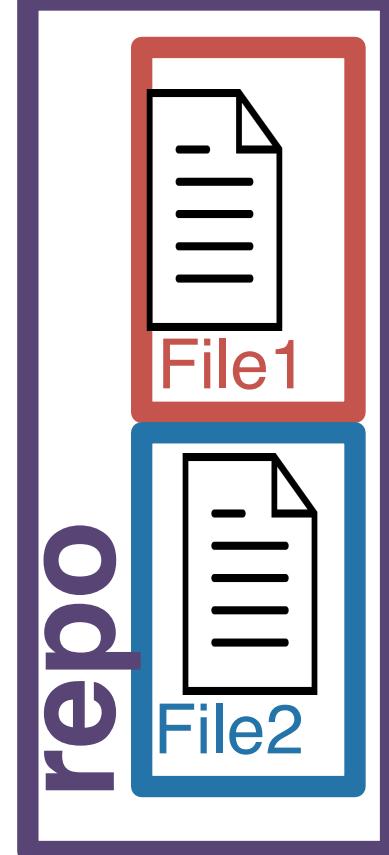


Shannon Ellis  
3/28/21 3:28pm

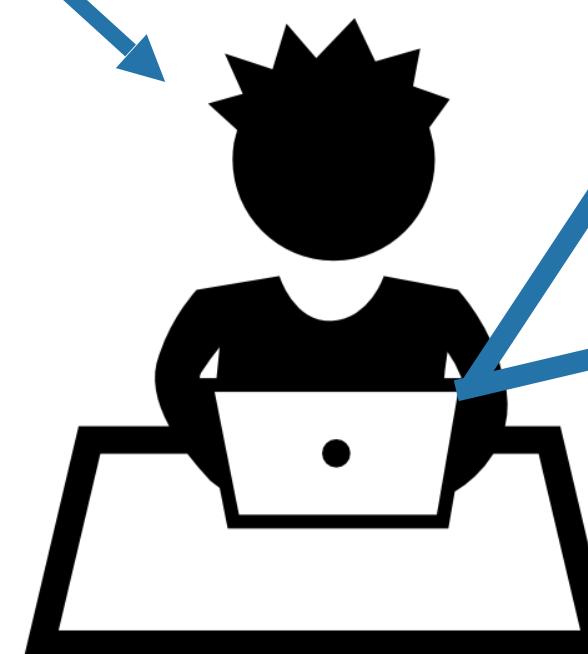
*fix typos in car and prof*



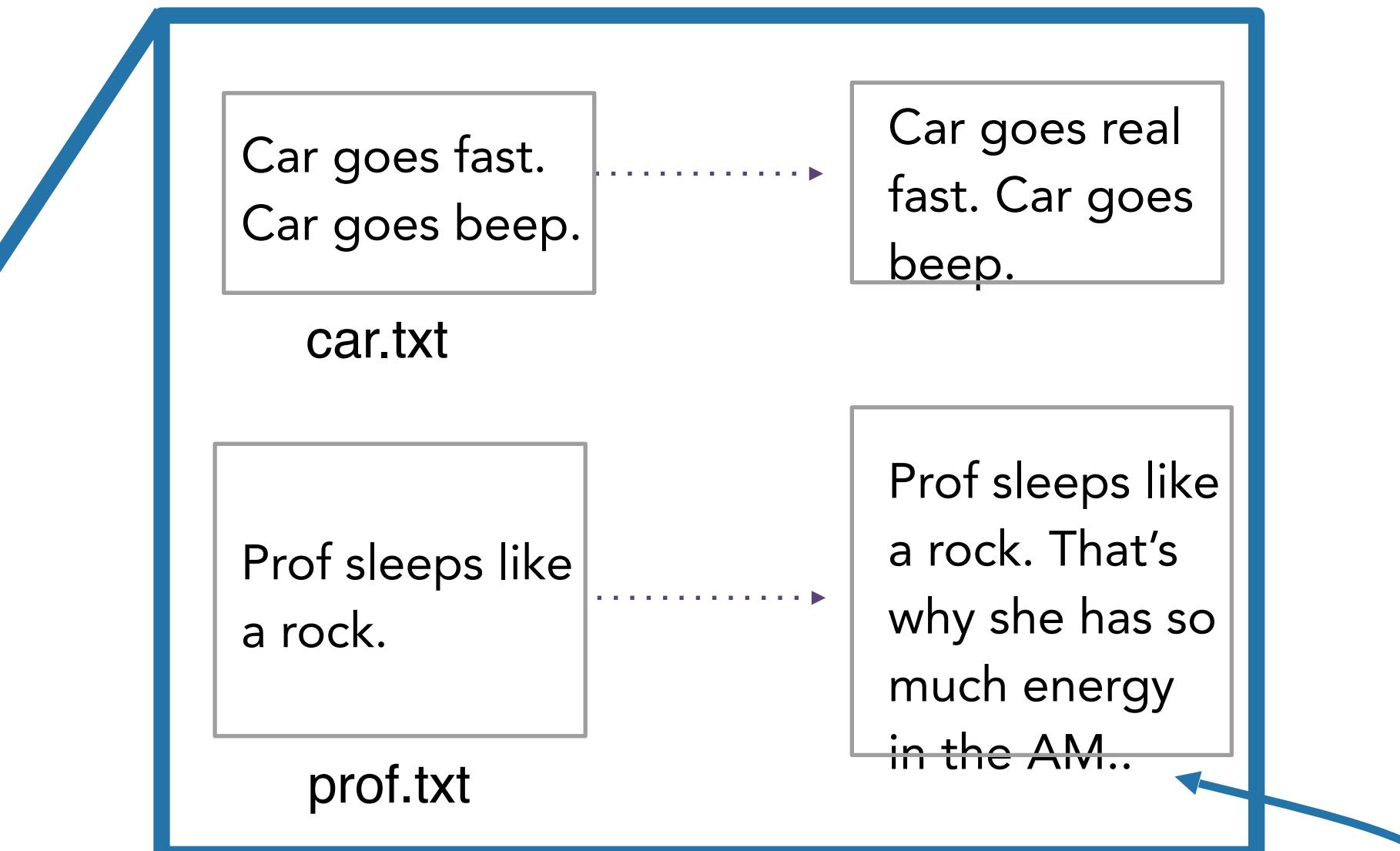
repo



*pull*



Your teammate pulls  
from remote and is  
now up-to-date!

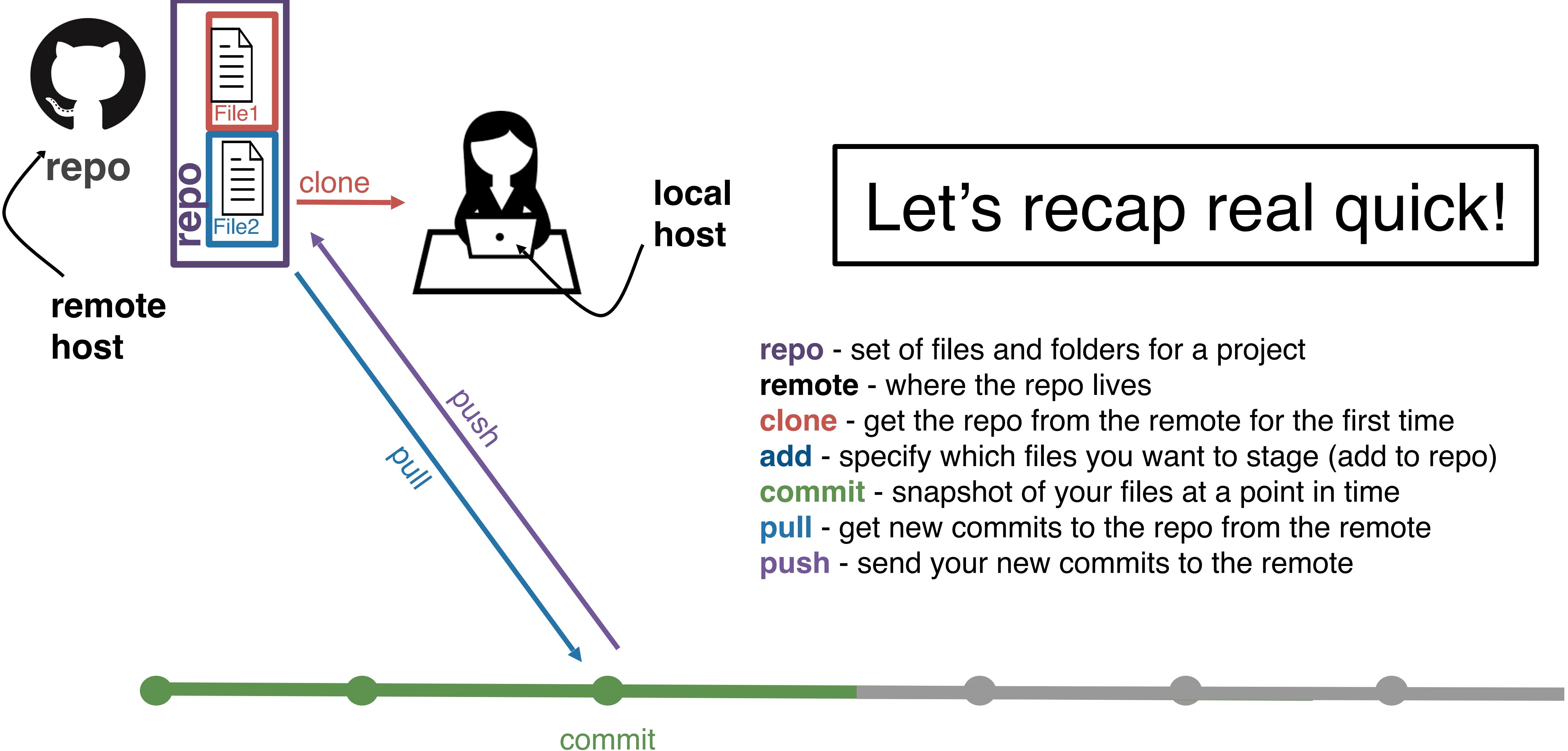


The files in his project  
locally will now have  
the updated files



Shannon Ellis  
3/28/21 3:28pm

*fix typos in car and prof*



## Let's recap real quick!

**repo** - set of files and folders for a project

**remote** - where the repo lives

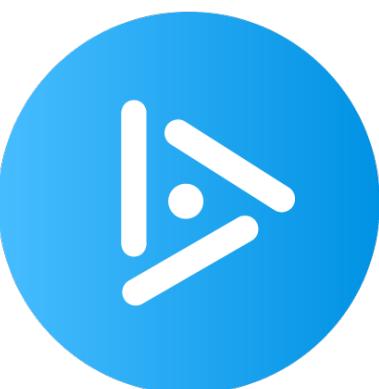
**clone** - get the repo from the remote for the first time

**add** - specify which files you want to stage (add to repo)

**commit** - snapshot of your files at a point in time

**pull** - get new commits to the repo from the remote

**push** - send your new commits to the remote

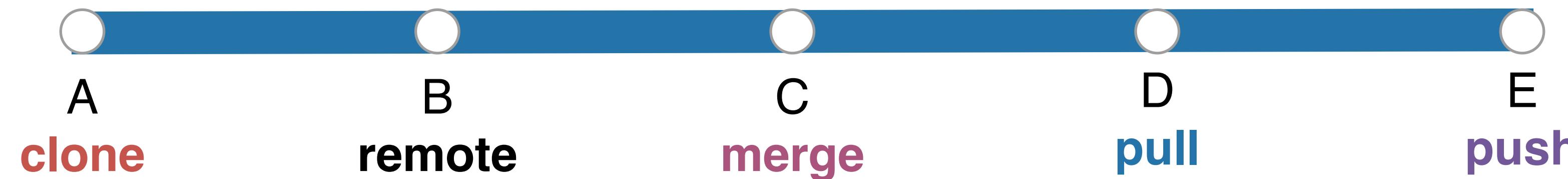


# Version Controller I

<https://forms.gle/wHA2GSyuycFre5qr6>

You've been working with a team on a project in a repo. You've made changes locally and you want to see them on the remote.

**What do you do to get them on the remote?**

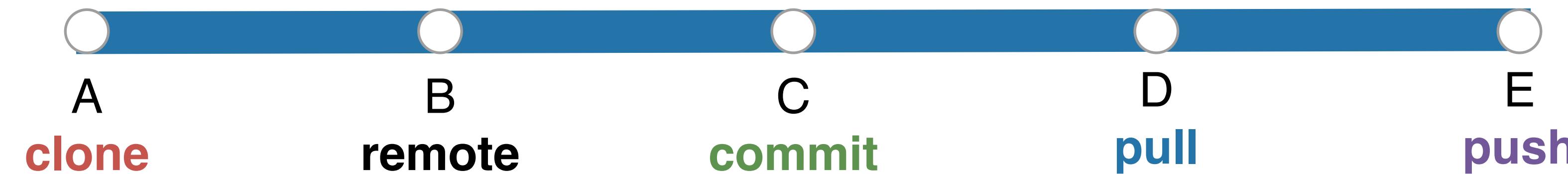




## Version Controller II

Your teammate has given you access to a GitHub repository to work on a project together. You want to get them for the first time on your computer locally.

**What do you do to get the repo on your computer?**





## Git exercise #1

1. Login to [datahub.ucsd.edu](https://datahub.ucsd.edu)

2. Open a terminal

3. In the terminal type:

```
git config --global user.email "you@ucsd.edu"
```

```
git config --global user.name "Your Name"
```

```
git clone https://github.com/COGS108/Lectures-Fa25
```

4. Ask yourself, how will the command look different when I want to update my lecture repo with the new slides next week. Try that command out!

## Git exercise #2 - updating a repo

1. Change directory to the new repo Lectures-Sp25
2. Figure out if what (if anything) is different from the last commit, type:

```
git status
```

3. Create a brand new file

```
touch 007-new-movie.mov
```

4. Figure out if what (if anything) is different from the last commit, type:

```
git status
```

5. Stage your changes

```
git add 007-new-movie.mov
```

6. Commit your changes

```
git commit -m "unreleased James Bond movie"
```

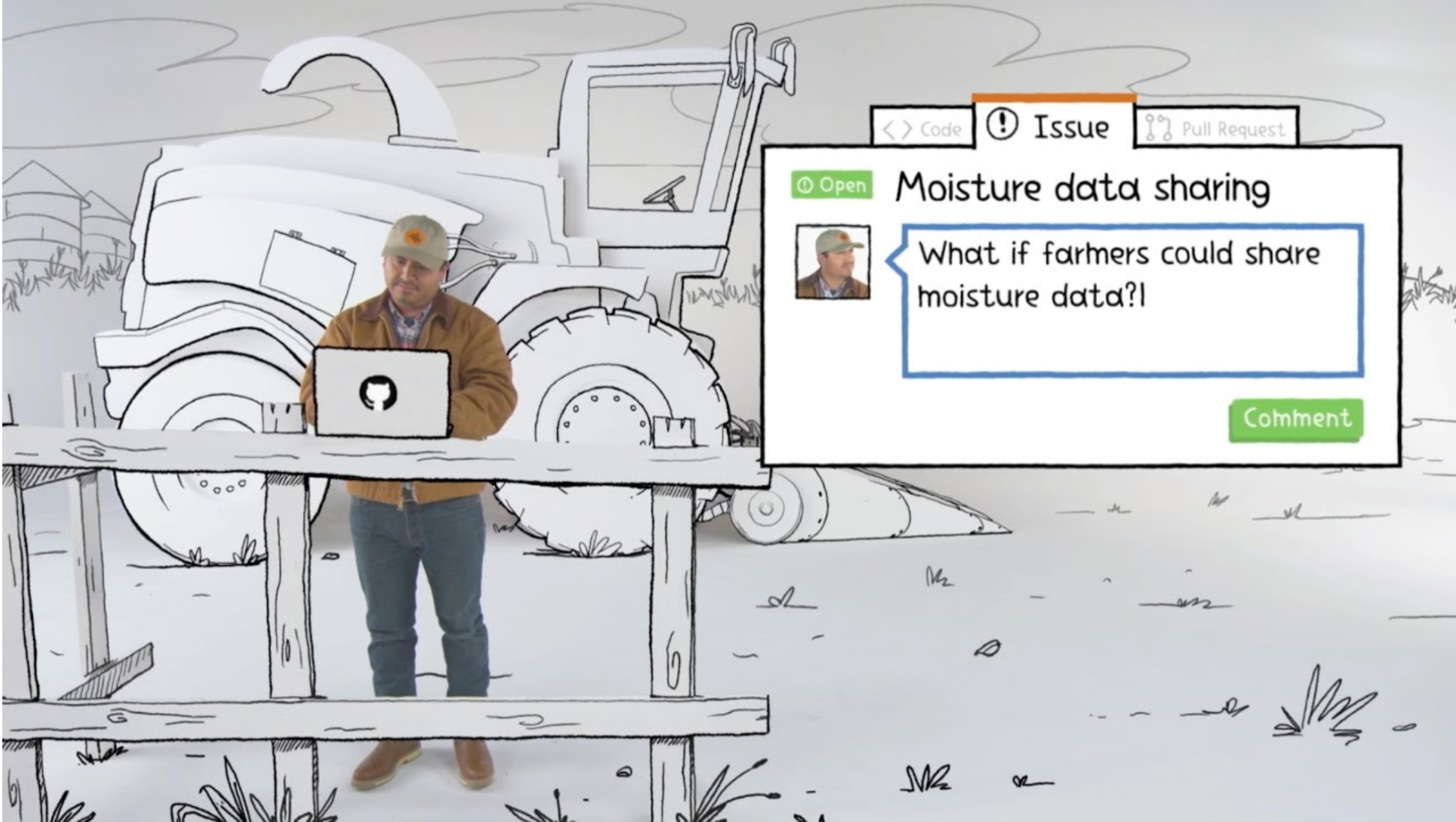
7. Question: What happens if you push to Github? Why?

8. Undo these unwanted changes to prevent problems in the future:

```
git reset --hard HEAD~1
```

Link for the video on the next slide so you can watch it on your own:

<https://youtube.com/watch?v=w3jLJU7DT5E>



< > Code

! Issue

Pull Request

Open



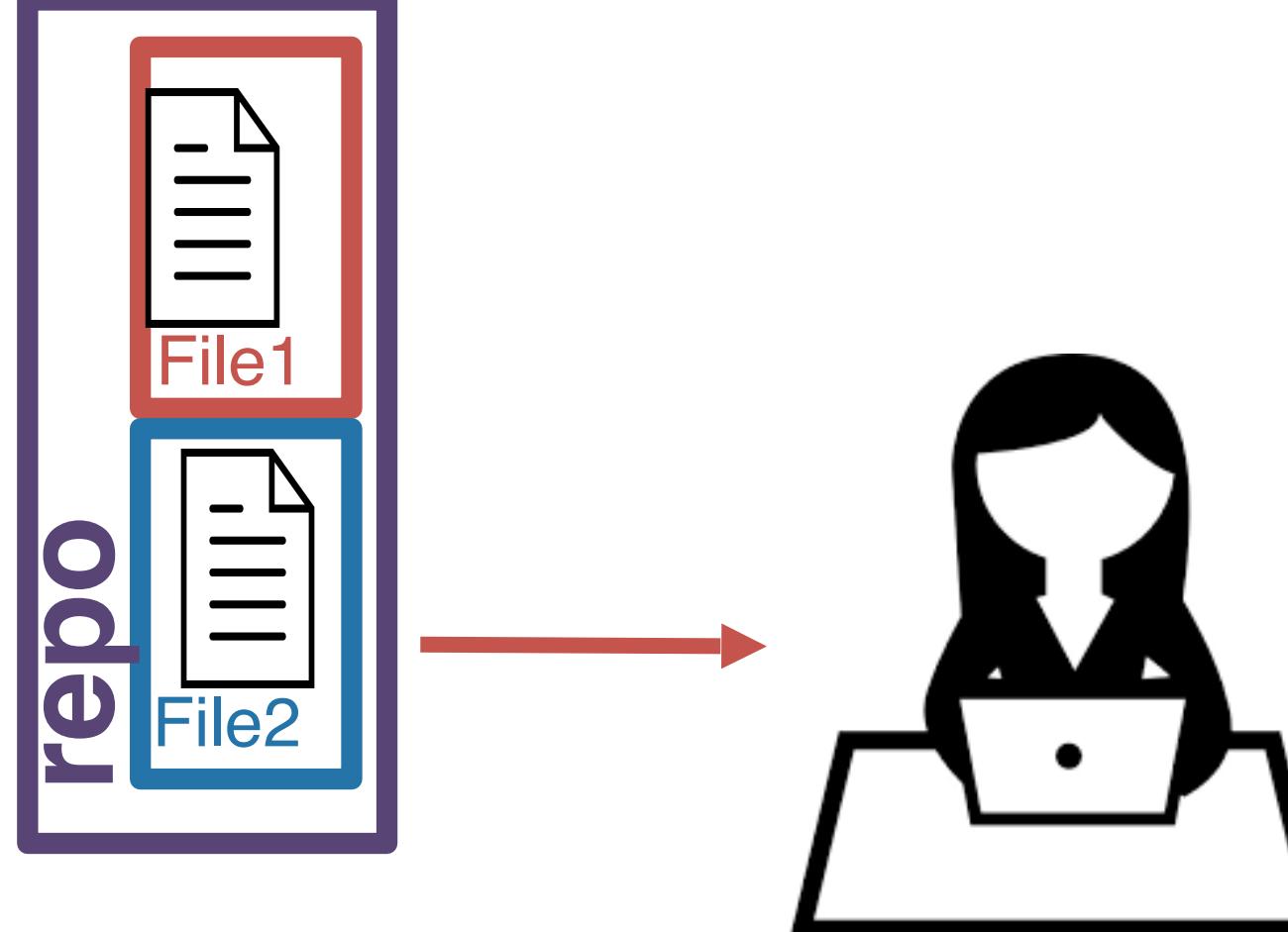
## Moisture data sharing

What if farmers could share moisture data?!

Comment

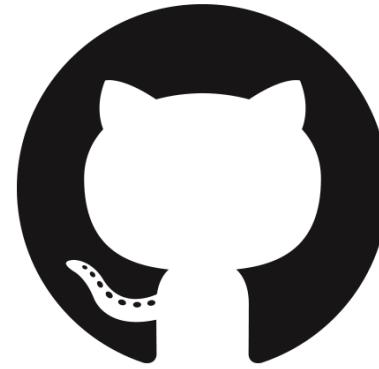


repo

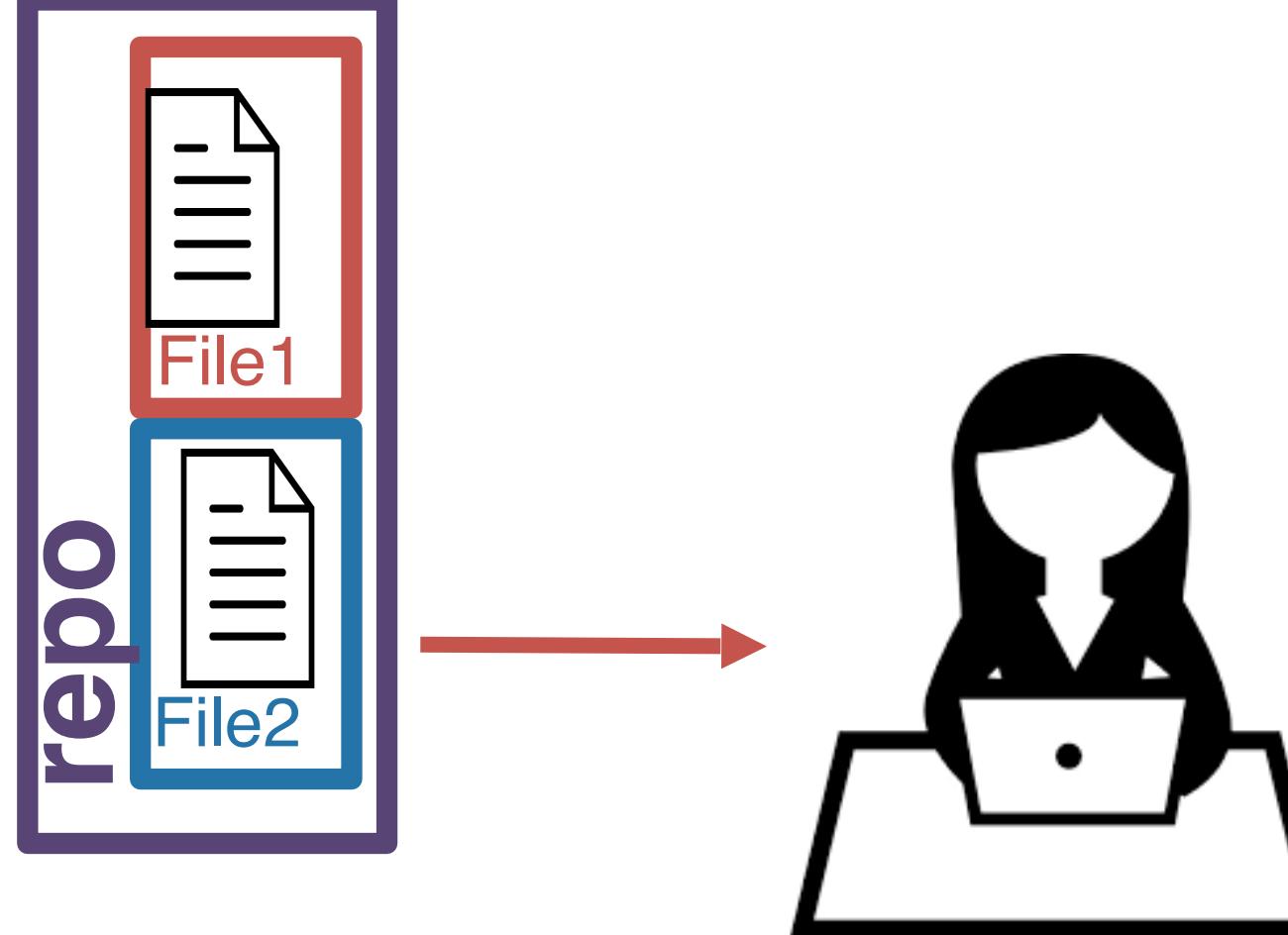


Each time you create a commit, git tracks  
the changes made automatically.



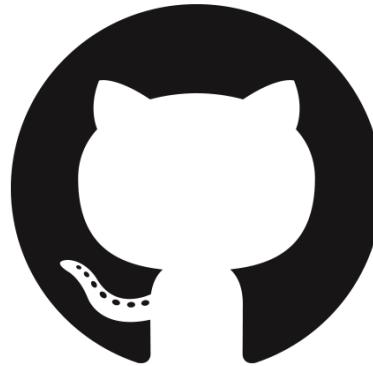


repo

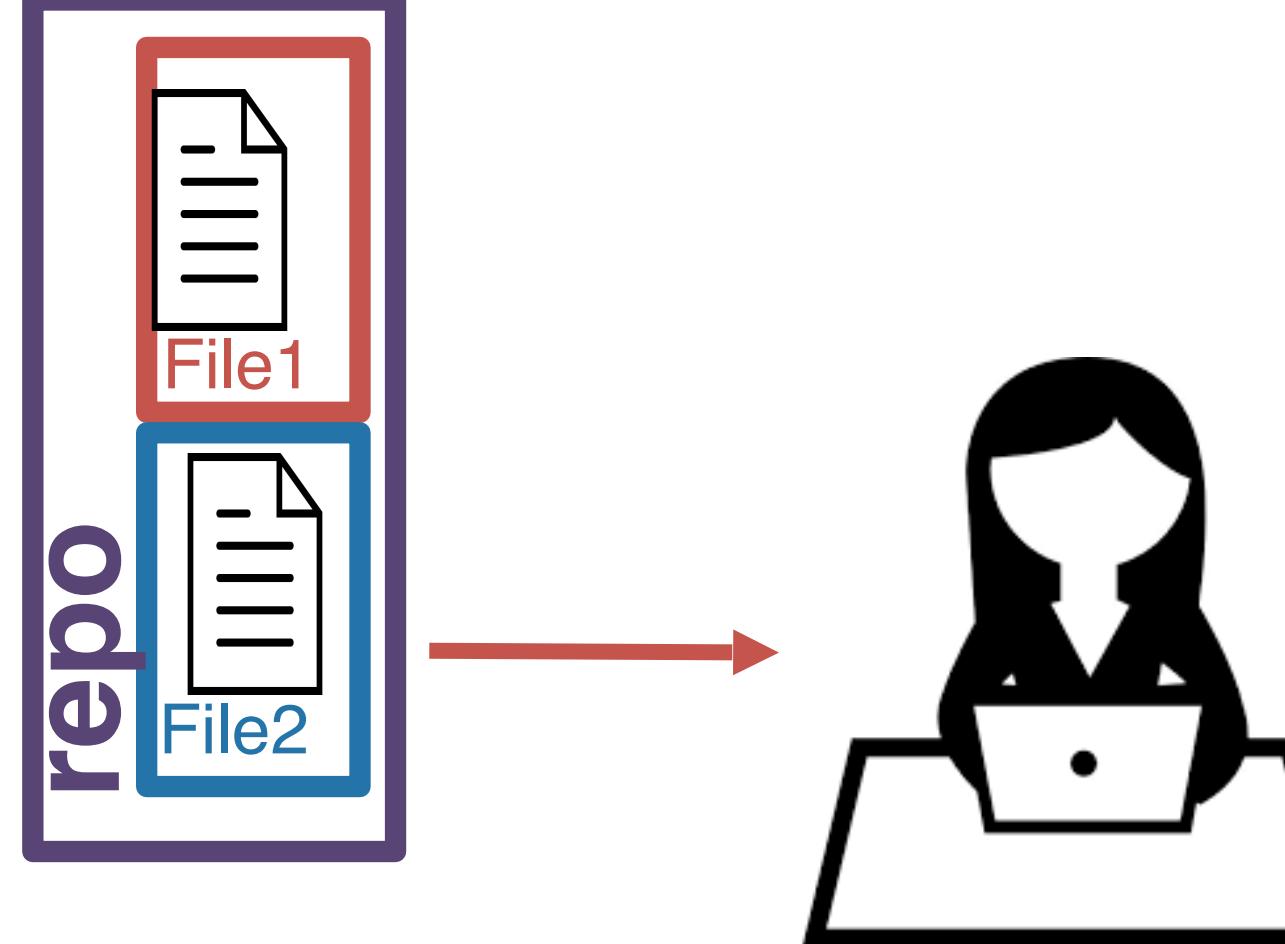


By committing each time you make changes, git allows you to time travel!





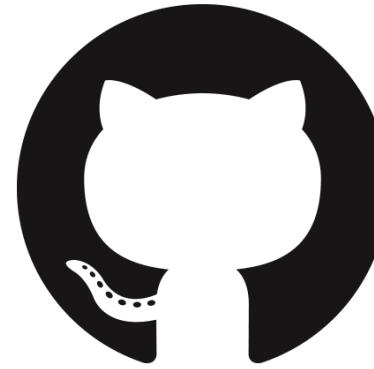
repo



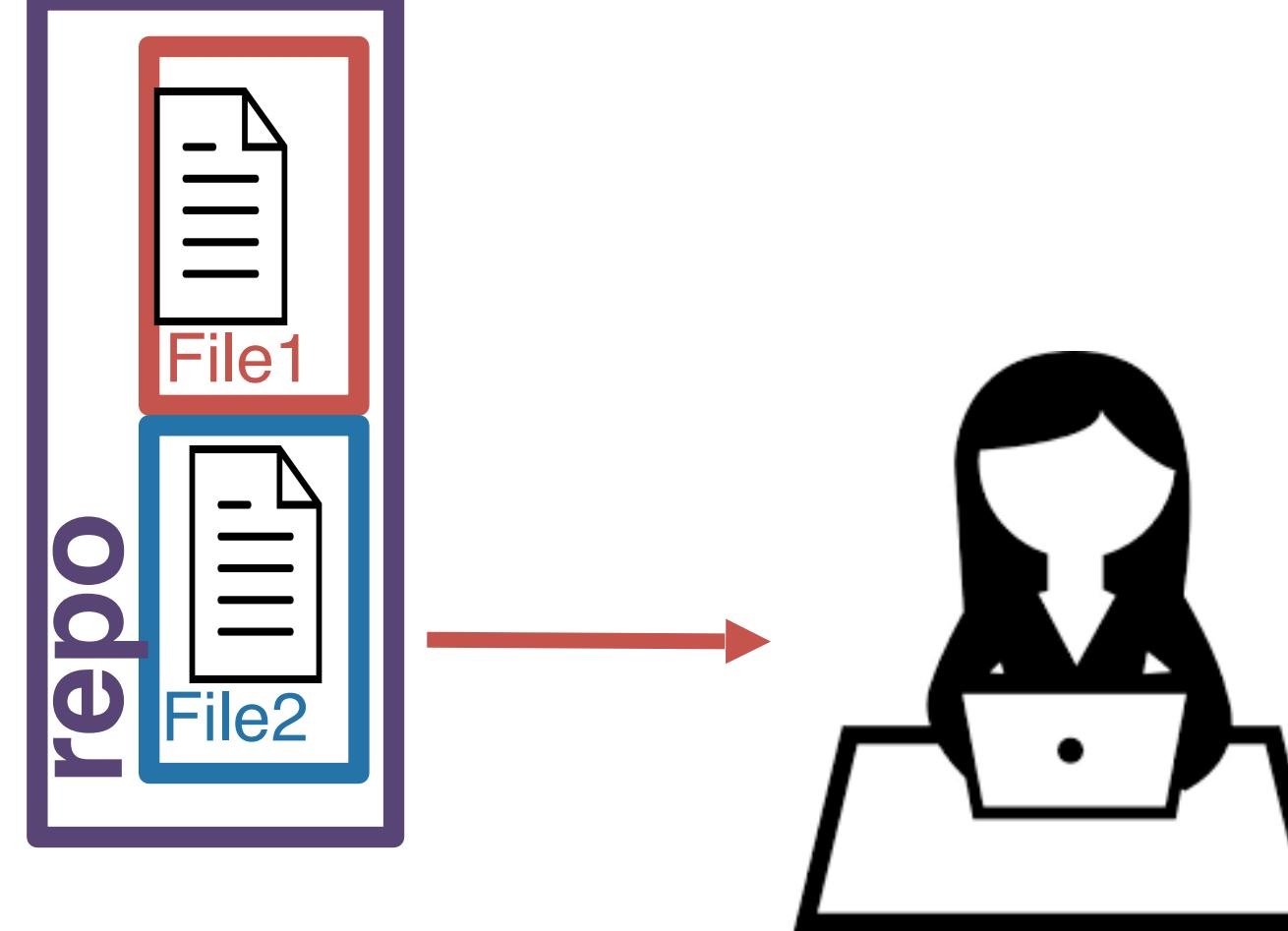
By committing each time you make changes, git allows you to time travel!



There's a unique id, known as a **hash**, associated with each commit.

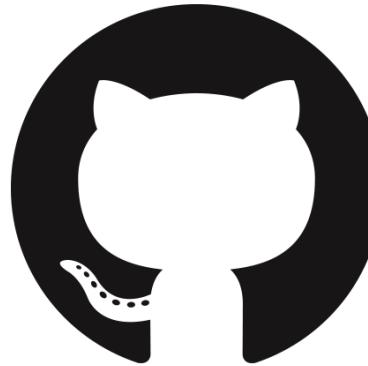


repo

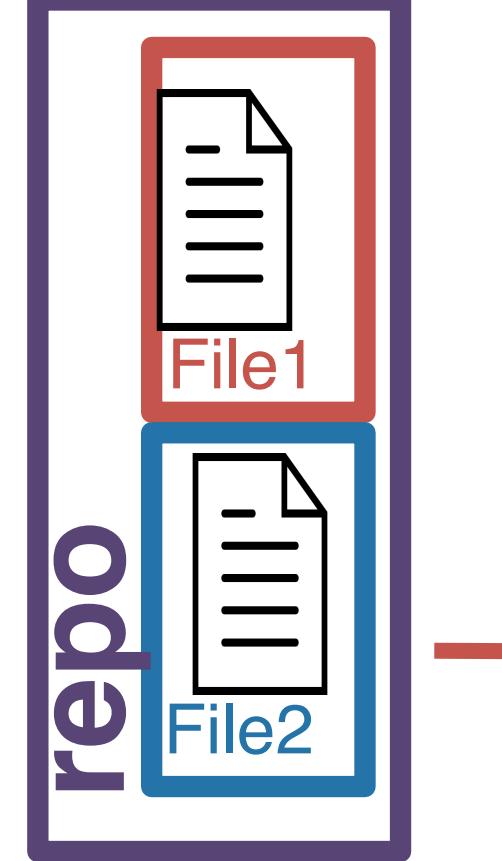


You can return to the state of the repository at any commit. Future commits don't disappear. They just aren't visible when you **check out** an older commit.



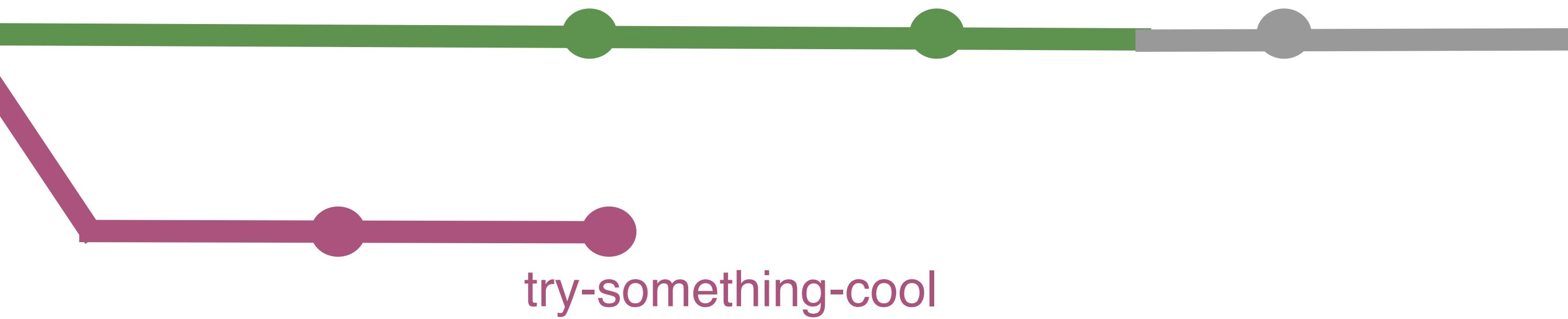


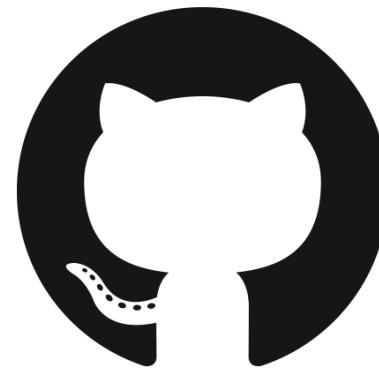
repo



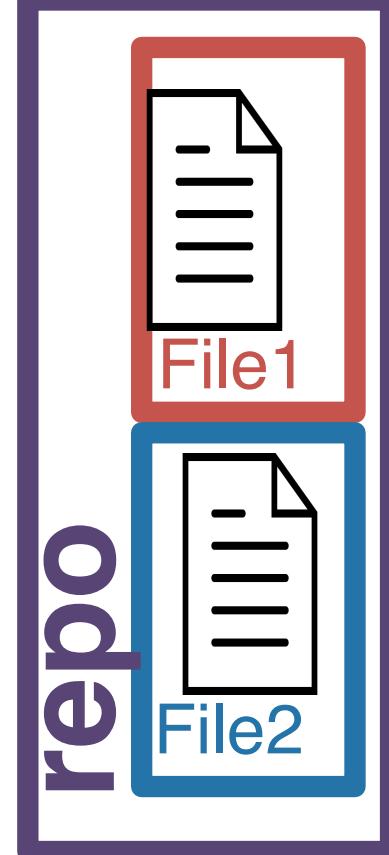
main branch

But...not everything is always linear.  
Sometimes you want to try something out  
and you're not sure it's going to work.  
This is where you'll want to use a **branch**.





repo



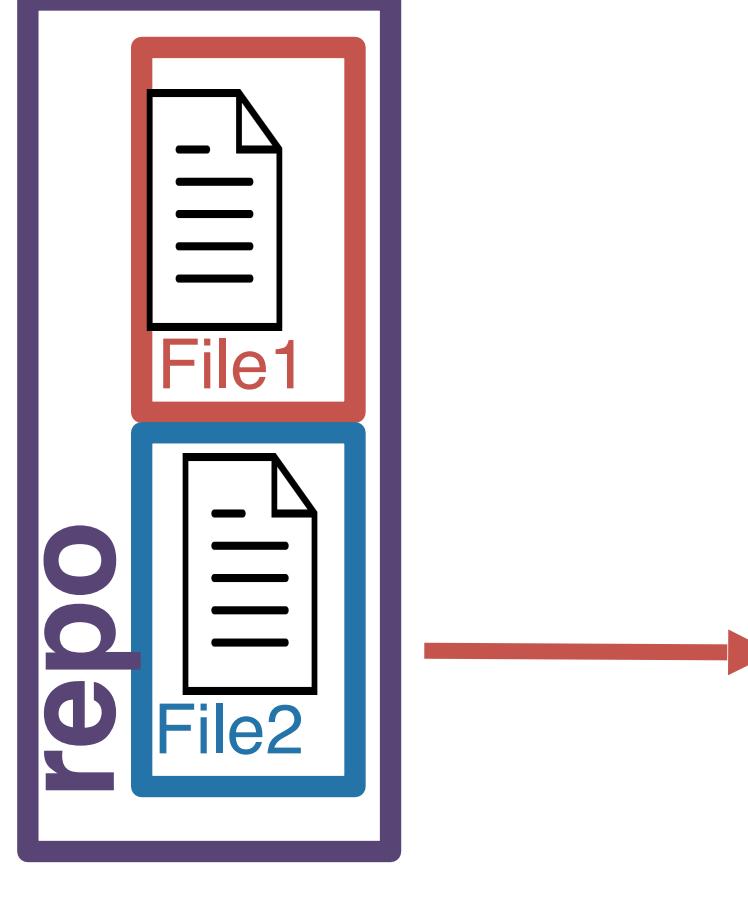
main branch

It's a good way to experiment. It's pretty easy to get rid of a branch later on should you not want to include the commits on that branch.

try-something-cool



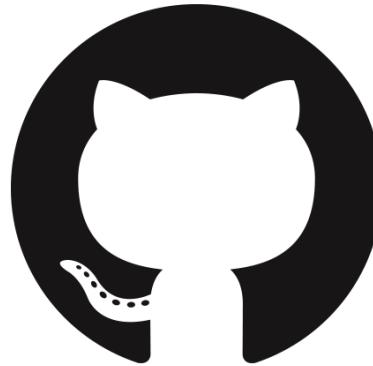
repo



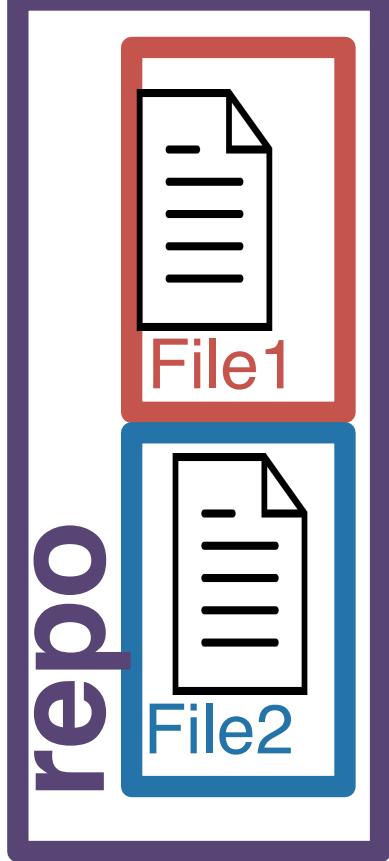
main

But...what if you DO want to include the changes you've made on your try-something-cool branch into the main branch?

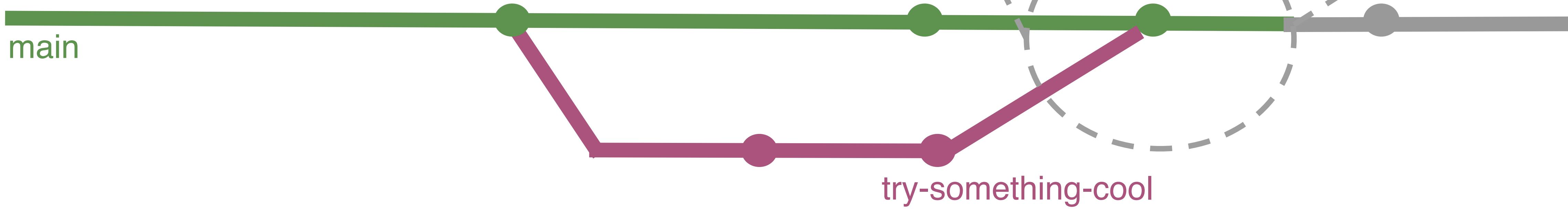
try-something-cool



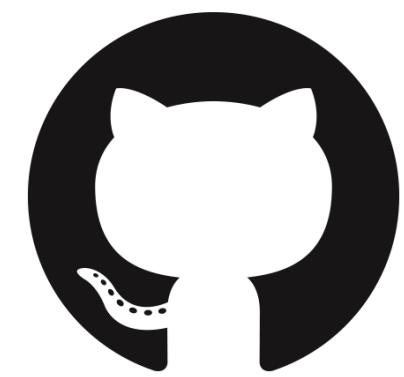
repo



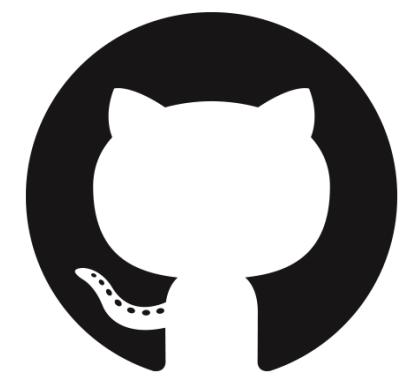
main



A merge allows you to combine the commits from a branch back into the main.

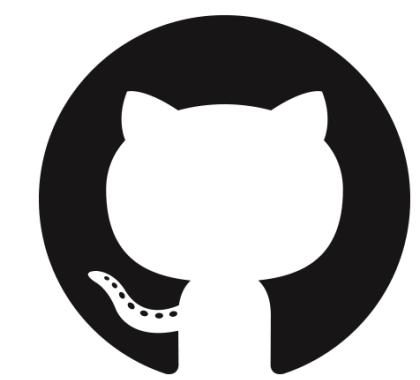


someone  
else's  
repo

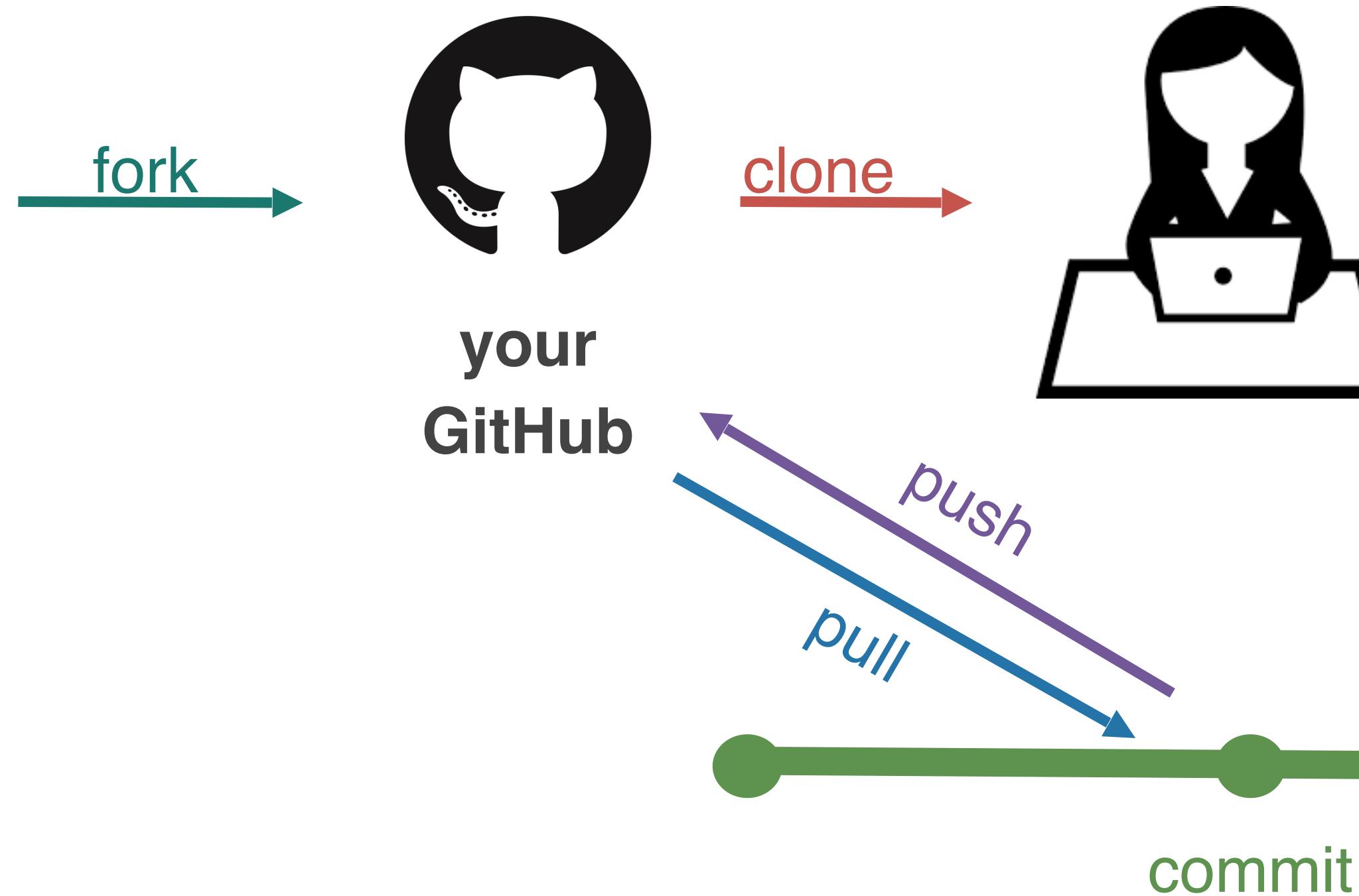


your  
GitHub

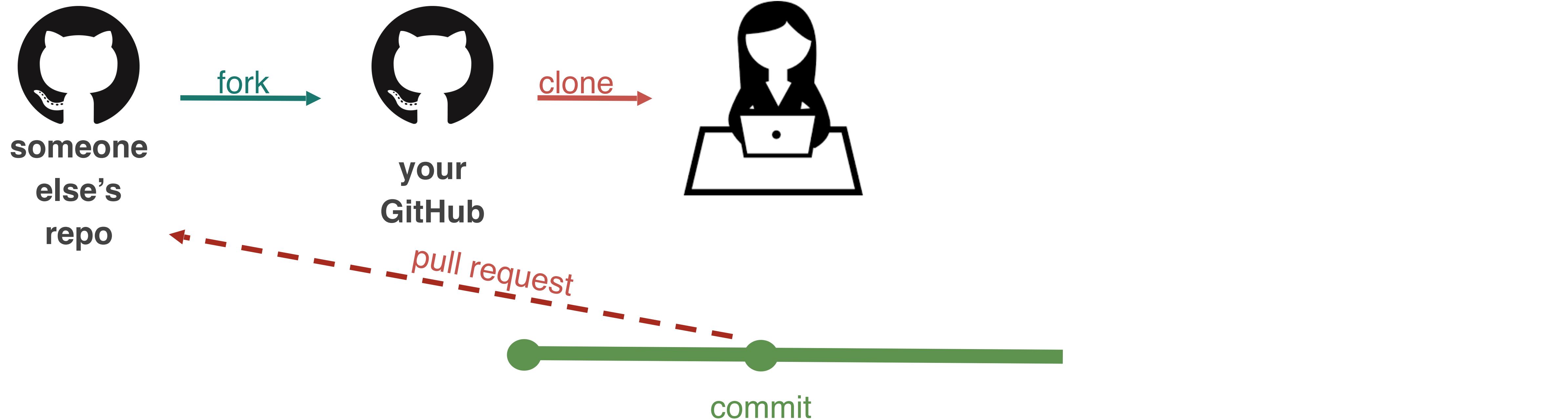
What if someone else is working  
on something cool and you want  
to play around with it? You'll  
have to **fork** their repo.



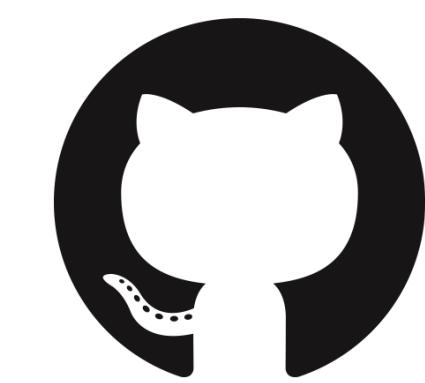
someone  
else's  
repo



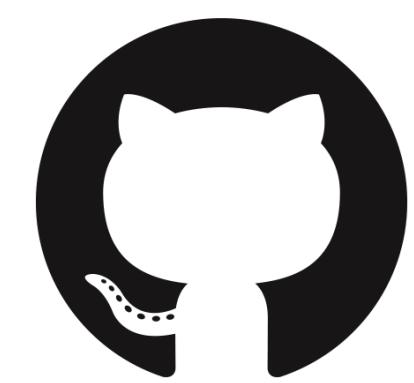
After you fork their repo, you can play around with it however you want, using the workflow we've already discussed.



But what if you think you've found a bug in their code, a typo, or want to add a new feature to their software? For this, you'll submit a **pull request** (aka **PR**).



someone  
else's  
repo



your  
GitHub



fork

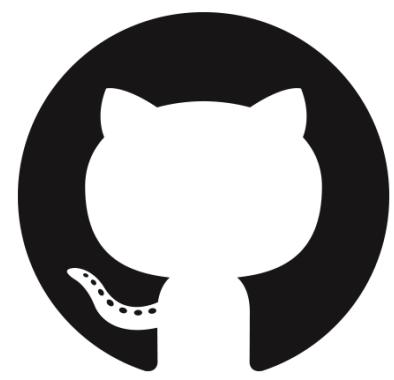
clone

The author then  
reviews your code/  
edits and decides  
whether or not they  
want to **merge your**  
**pull request**.

*pull request*

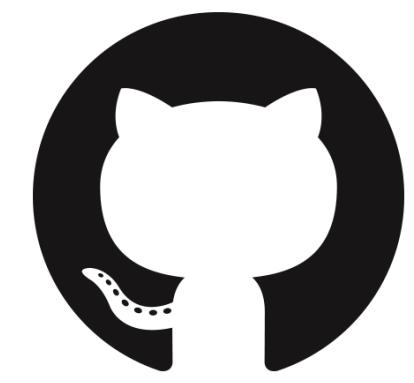
commit

But what if you think you've found a bug in  
their code, a typo, or want to add a new  
feature to their software? For this, you'll  
submit a **pull request** (aka **PR**).



someone  
else's  
repo

Last but not least...what if you find a bug in someone else's code OR you want to make a suggestion but aren't going to submit a suggestion with a PR. For this, you can file an **issue** on GitHub.



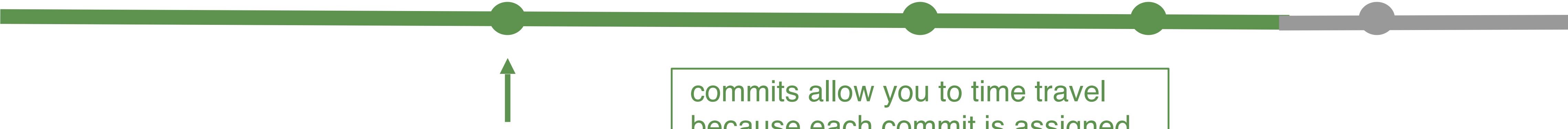
someone  
else's  
repo

Last but not least...what if you find a bug in someone else's code OR you want to make a suggestion but aren't going to submit a suggestion with a PR. For this, you can file an **issue** on GitHub.

**Issues** are *bug trackers*. While, they can include bugs, they can also include feature requests, to-dos, whatever you want, really!

They can be assigned to people.

They can be closed once addressed ....or if the software maintainer doesn't like the suggestion



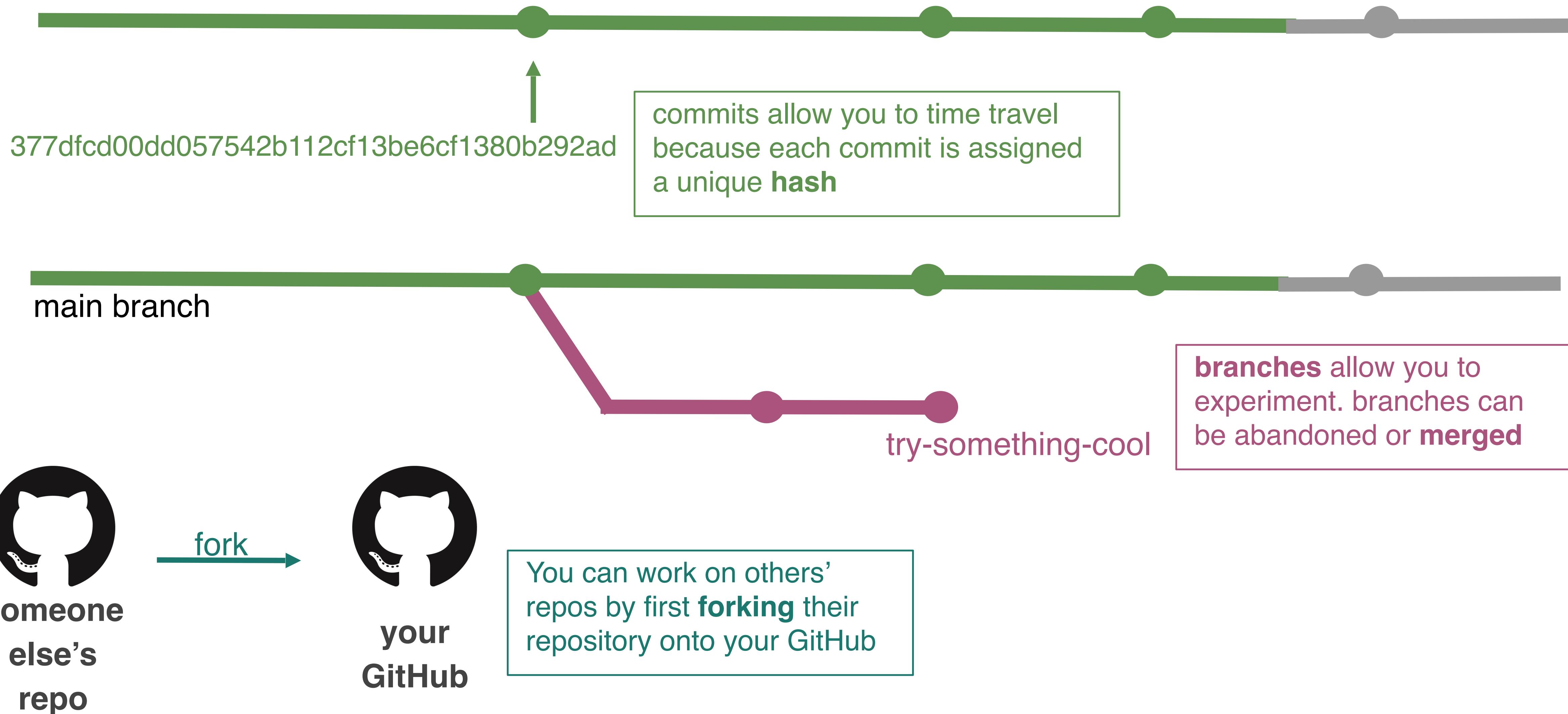
377dfcd00dd057542b112cf13be6cf1380b292ad

commits allow you to time travel  
because each commit is assigned  
a unique **hash**

One more git recap...



One more git recap...



One more git recap...

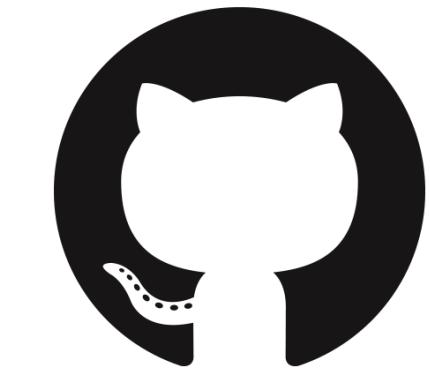
377dfcd00dd057542b112cf13be6cf1380b292ad

commits allow you to time travel because each commit is assigned a unique **hash**

main branch

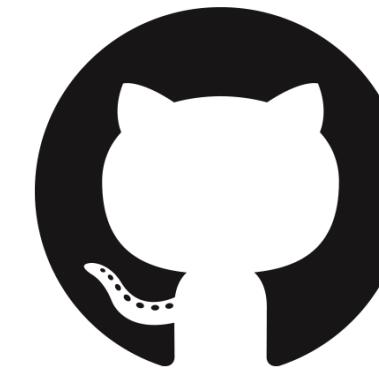
try-something-cool

**branches** allow you to experiment. branches can be abandoned or **merged**



someone  
else's  
repo

fork



your  
GitHub

You can work on others' repos by first **forking** their repository onto your GitHub

One more git recap...

**Pull requests** allow you to make specific edits to others' repos

**Issues** allow you to make general suggestions to your/others' repos

**Review & Question Time**

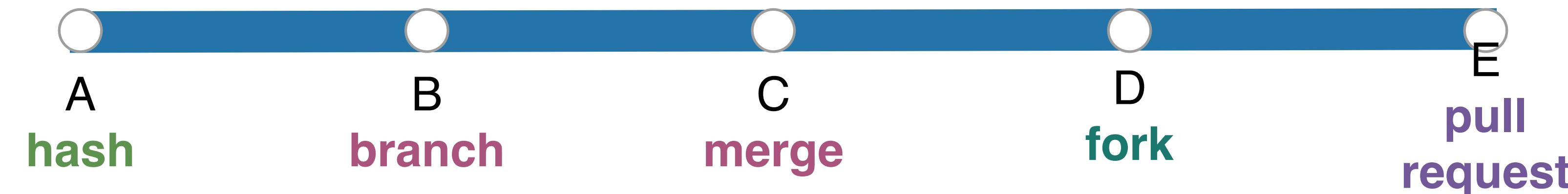


# Version Controller III

<https://forms.gle/eyxgHB3wvqmy17uR9>

To experiment within your own repo (test out a new feature, make some changes you're not sure will work)...

what should you do?

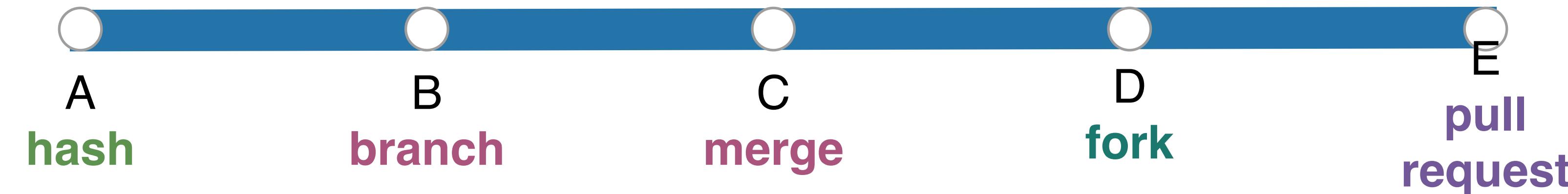




# Version Controller IV

<https://forms.gle/eyxgHB3wvqmy17uR9>

If you've made edits to someone else's repo that you're not a collaborator on...  
what would *they* have to do to incorporate your changes?







# Good to Git

Use it for...

- Plain text files
- < 100MB per file limit
- Total repository size < 5 GB



# Careful

Easy to mess up

- Metadata rich text files that change over time, e.g.
  - Jupyter Notebook .ipynb
  - Word processor formats like .docx
  - JSON data



# Don't Git

Avoid using for

- Large text files  648MB
- Vast directory structures  36,142,087 small text files in a nested subdirectory tree up to 14 levels deep
- Binary files  .jpg, .mp4, .sql, etc



## Git LFS can help!

# Version Control: Practice

- GO TO GITHUB AND GET A USERNAME! We need it for many things in the class
- You will use git & Github when you do
  - Discussion Lab 1
  - Assignment 1
- Understand what you're doing in the assignment!
- You may have to google, ask others, spend some time with this!
- git & Github == How to get the course lectures/materials
  - Assignment 1 will have you fork the Lectures and Project repos
  - You can keep the lectures up-to-date throughout the quarter
- you'll be using GitHub for your final projects
- Fill in the quiz before the end of week 2 so we have your username for creating project repos!

Week 2

- Q1 Oct 9 | 1 pts
- Practice Assignment Oct 11 | 1 pts
- D1 Oct 13 | 2 pts
- Github Username Oct 13

# Using SSH to contribute work to your project repo

**You can't write or access private repos without authentication!**

Click here, follow instructions:

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account>

The screenshot shows a dark-themed web page from GitHub's documentation. At the top, it says 'Authentication / Connect with SSH /'. Below that, the main title is 'Adding a new SSH key to your GitHub account'. A sub-instruction reads: 'To configure your account on GitHub.com to use your new (or existing) SSH key, you'll also need to add the key to your account.' There are navigation links for 'Mac', 'Windows', 'Linux', 'GitHub CLI', and 'Web browser' (which is underlined). Under 'About addition of SSH keys to your account', it says: 'You can access and write data in repositories on GitHub using SSH (Secure Shell Protocol). When you connect via SSH, you authenticate using a private key file on your local machine. For more information, see [About SSH](#). You can also use SSH to sign commits and tags. For more information about commit signing, see [About commit signature verification](#). After you generate an SSH key pair, you must add the public key to GitHub.com to enable SSH access for your account.'

You will need to do this for each local computer you will use  
(e.g., on Datahub and again on your laptop!)

# GitFu level: advanced

Installing git

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

Installing GH GLI

<https://cli.github.com/>

Other options include Github Desktop, Source Tree, VSCode, or any other IDE

GO TO GITHUB

# Jupyter notebooks suck to version control

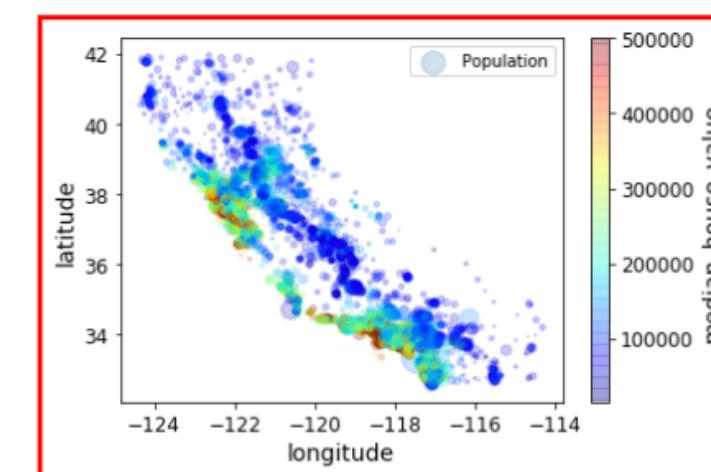
<https://nextjournal.com/schmudde/how-to-version-control-jupyter>

## ReviewNB

ReviewNB is a GitHub app that also offers visual diffing with an interface that looks similar to the traditional Jupyter IDE. Because the outputs are visualized, problems associated with committing binary blobs disappear.

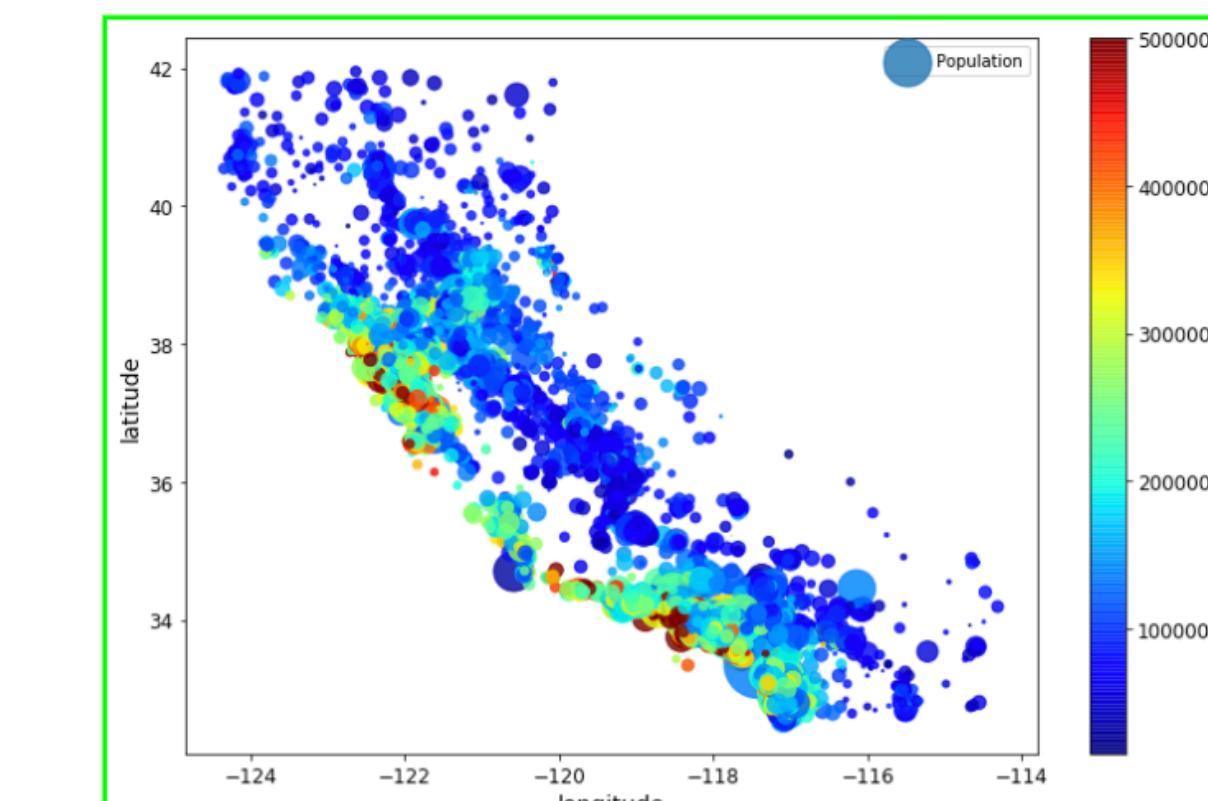
```
1 housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.2,
2     s=housing["population"]/88, label="Population", figsize=(6,4),
3     c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
4     sharex=False)
5 plt.legend()
6 save_fig("housing_prices_scatterplot")
```

Saving figure housing\_prices\_scatterplot



```
1 housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.8,
2     s=housing["population"]/21, label="Population", figsize=(10,7),
3     c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
4     sharex=False)
5 plt.legend()
6 save_fig("housing_prices_scatterplot")
```

Saving figure housing\_prices\_scatterplot



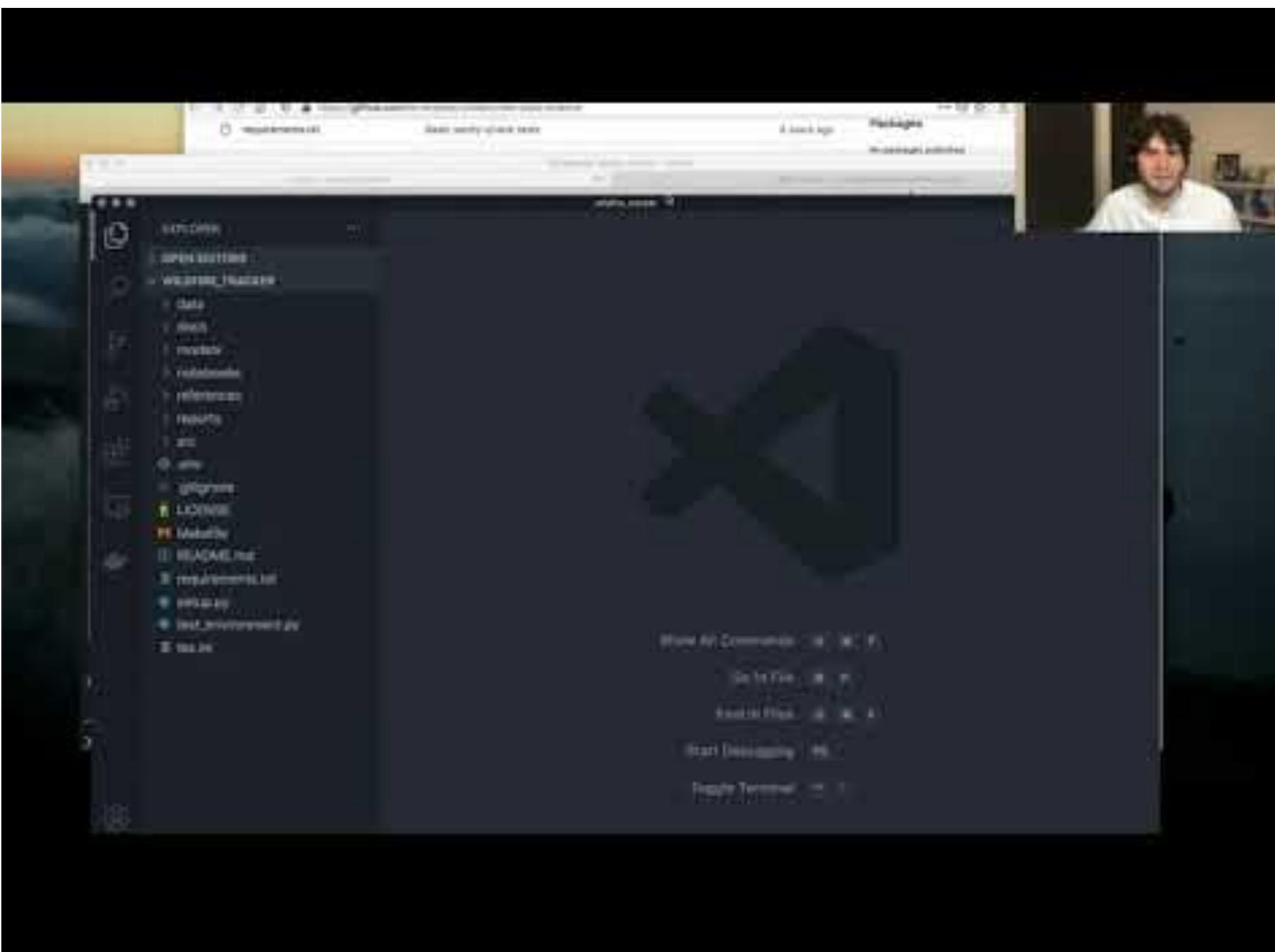
ReviewNB example courtesy of the [ReviewNB website](#)

# More options

nbautoexport

docs stable | pypi v0.5.0 | conda-forge v0.5.0 | tests passing | codecov 99%

Making it easier to code review Jupyter notebooks, one script at a time.



- jupy  
+ text

**Using text notebooks**

Demo

- example.py has a notebook icon! Open it as a notebook
- Set the appropriate kernel and run the notebook
- Modify the file in Jupyter, save, and see the change on the script
- Modify the script in a text editor (1), and reload it in Jupyter:
  - Inputs are updated
  - Outputs are gone (2)
  - Python variables are still there
- Add the kernel to the .py file with the include Metadata command

(1) If your edit takes more than 2 minutes, answer 'Reload' to the message in Jupyter telling you that the notebook has changed on disk, and consider a) turning the Jupyter autosave off or b) closing the notebook while you edit the text file

(2) To be solved at the next slide

# Lets practice!

**At least if we have enough time right now :)**

- Create repo
- Create file.md
- Create a branch “my\_work”
- On “my\_work”: Add a new file, prog.py
- On “my\_work”: Change contents of file.md
- On “master”: Change contents of file.md in a way that conflicts with “my\_work”
- Merge “my\_work” into “master”
- Resolve conflicts, finish merge

# Practice

You need to learn by doing. Listening to me blah blah is only going to go so far. Here's some choices that seem to me like they would be good for beginner to intermediate levels, but note I have not actually used them.

- Free 16 hour Coursera course by Google covering common Git usage patterns <https://www.coursera.org/learn/introduction-git-github>
- Katas for Git <https://github.com/eficode-academy/git-katas>
- If you have other suggestions please let me know!
- If you use these and like or hate them please give me feedback!