Version Control

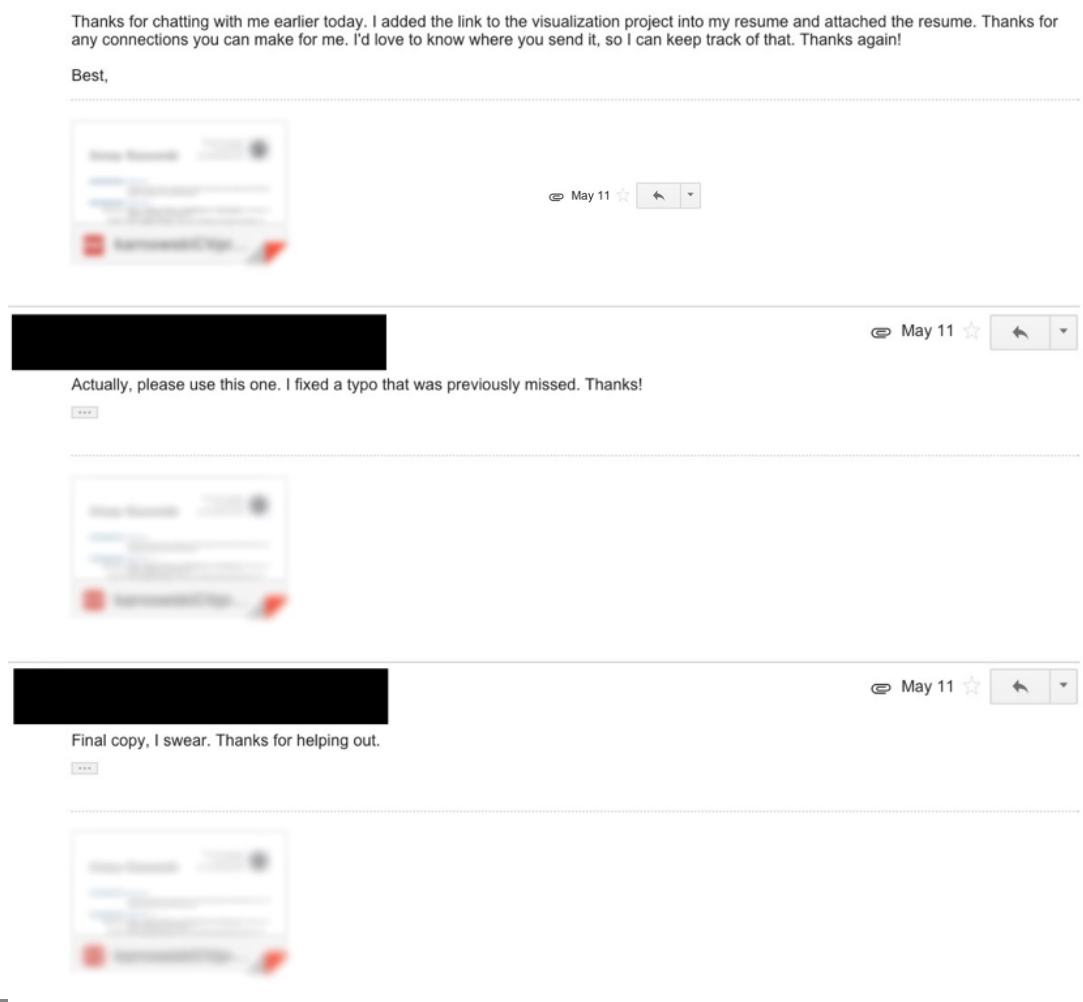
C. Alex Simpkins Jr., Ph.D
RDPRobotics LLC,
UC San Diego, Department of Cognitive Science
rdprobotics@gmail.com
csimpkinsjr@ucsd.edu

Lectures :http://casimpkinsjr.radiantdolphinpress.com/pages/cogs108_ss1_23/index.html

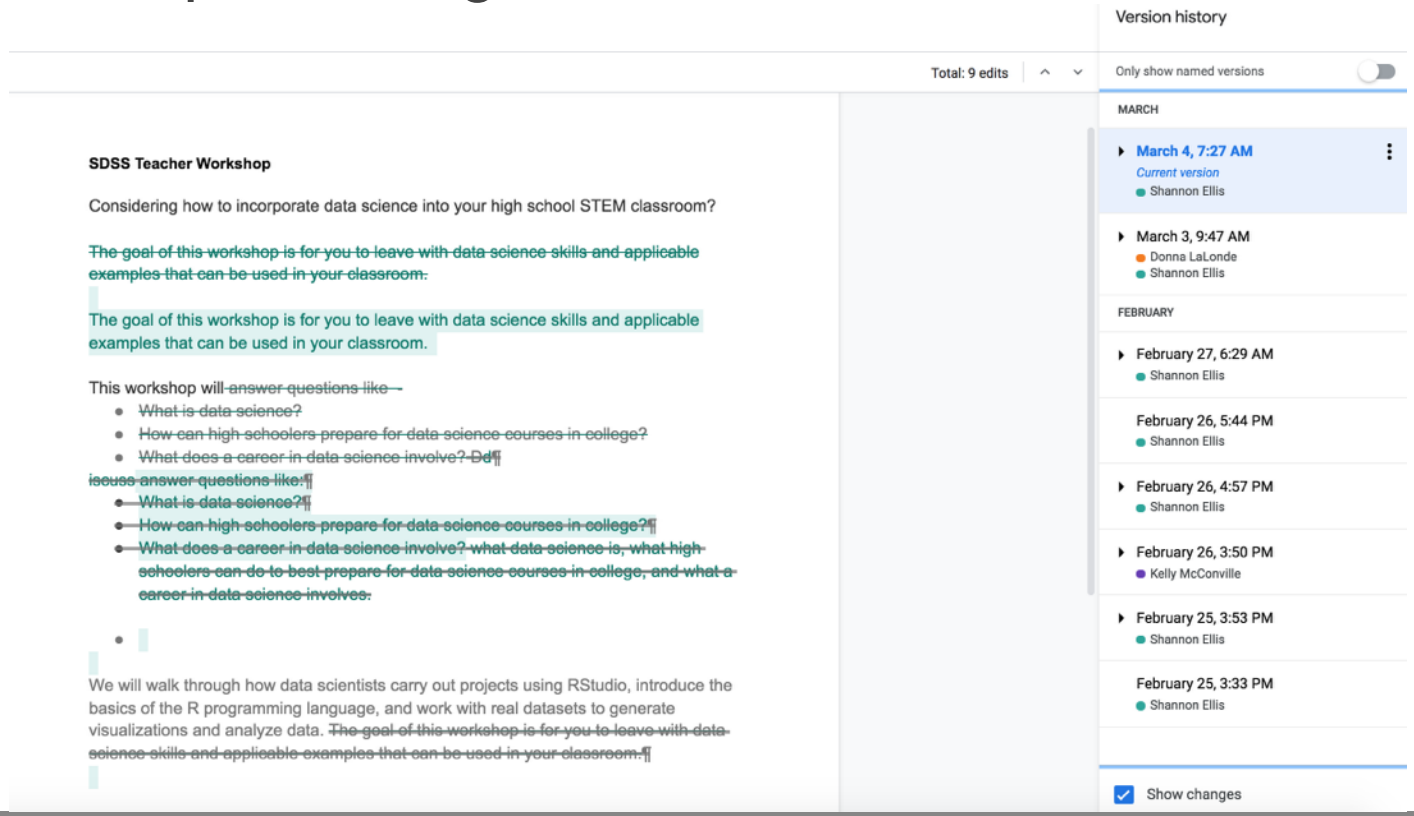
This sucks

main_simple_bak9-pretty-good.c	Aug 1, 2008, 1:01 AM	33 KB	C Source
main_simple_bak9-pretty-good.o	Aug 1, 2008, 1:00 AM	303 KB	object code
main_simple_bak9-pretty-goodv2.c	Aug 2, 2008, 1:16 AM	33 KB	C Source
main_simple_bak10.c	Sep 28, 2008, 1:16 PM	33 KB	C Source
main_simple_bak11-workingUART_correctspeed.c	Aug 30, 2008, 2:49 AM	27 KB	C Source
main_simple_bak11-workingUART_correctspeed.o	Aug 2, 2008, 1:17 AM	303 KB	object code
main_simple_bak12_willspin.c	Aug 2, 2008, 1:30 AM	28 KB	C Source
main_simple_bak12_willspin.o	Aug 2, 2008, 2:35 AM	301 KB	object code
main_simple_bak13-worksA-D-nonoise-spins.c	Aug 7, 2008, 12:57 PM	26 KB	C Source
main_simple_bak14-widersinefunctionsworkingrotation.c	Aug 8, 2008, 5:02 PM	26 KB	C Source
main_simple_bak15-spins-stillneedsquadrantfixed.c	Aug 15, 2008, 7:32 PM	30 KB	C Source
main_simple_bak16-15backup-spins-needs-improvement.c	Oct 15, 2008, 8:54 PM	31 KB	C Source
main_simple_bak17-smoother-stillnostandingstart.c	Aug 16, 2008, 6:50 PM	30 KB	C Source
main_simple_bak17-smoother-stillnostandingstart.o	Aug 18, 2008, 9:41 PM	305 KB	object code
main_simple_bak18-notgood.c	Aug 18, 2008, 9:42 PM	31 KB	C Source
main_simple_bak20SIMPLE-DCnotbrushless.c	Sep 17, 2009, 11:02 PM	27 KB	C Source
main_simple_bak20WORKS_PWM_COMMAND_CONTROL.c	Aug 19, 2008, 12:54 AM	29 KB	C Source
main_simple_timer_intrpt_bak.c	Aug 12, 2008, 12:16 AM	13 KB	C Source
main_simple_timer_intrpt_bak2.c	Aug 12, 2008, 2:00 PM	13 KB	C Source
main_simple_timer_intrpt_bak3.c	Aug 18, 2008, 12:14 AM	13 KB	C Source
main_simple_timer_intrpt.c	Aug 18, 2008, 12:17 AM	13 KB	C Source
main_simple_workingHWPWM.c	Aug 18, 2008, 7:19 PM	15 KB	C Source
main_simple.c	Sep 17, 2009, 11:02 PM	29 KB	C Source

Yup, this sucks too.



This is a step in the right direction



Version Control

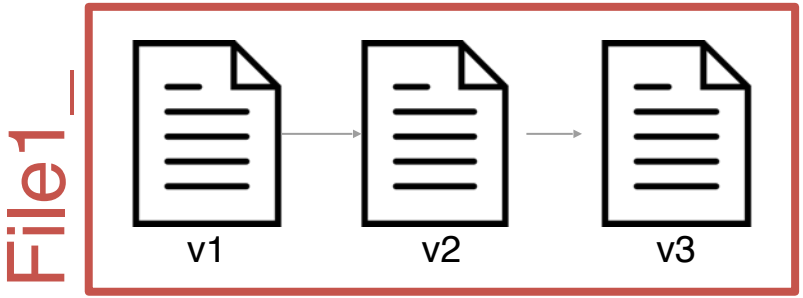
- Enables multiple people to simultaneously work on a single project.
- Each person edits their own copy of the files and chooses when to share those changes with the rest of the team.
- Thus, temporary or partial edits by one person do not interfere with another person's work

What is version control?

A way to manage the evolution of a set of files

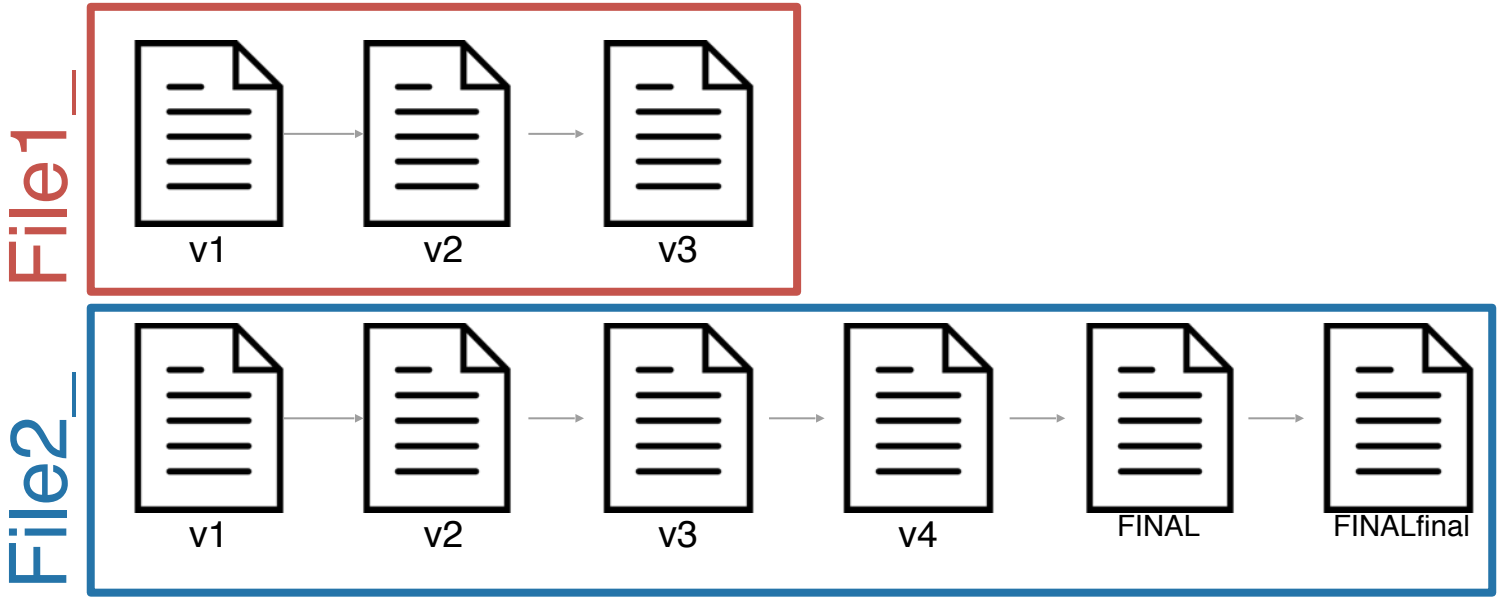
What is version control?

A way to manage the evolution of a set of files



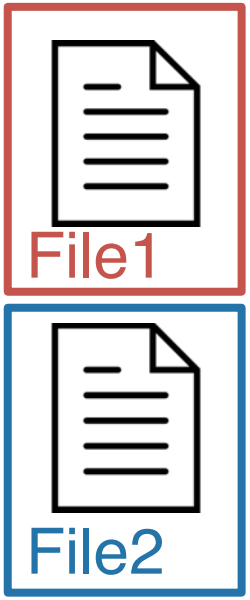
What is version control?

A way to manage the evolution of a set of files



What is version control?

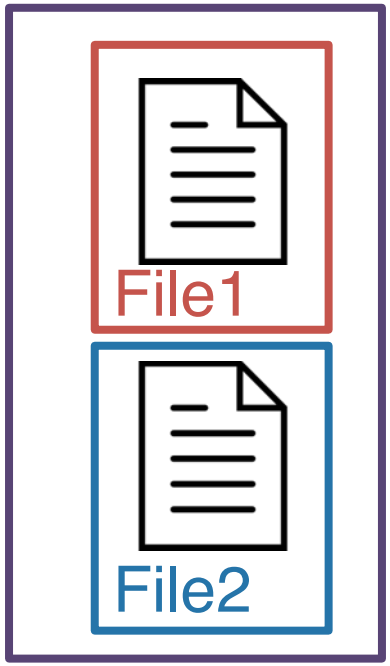
A way to manage the evolution of a set of files



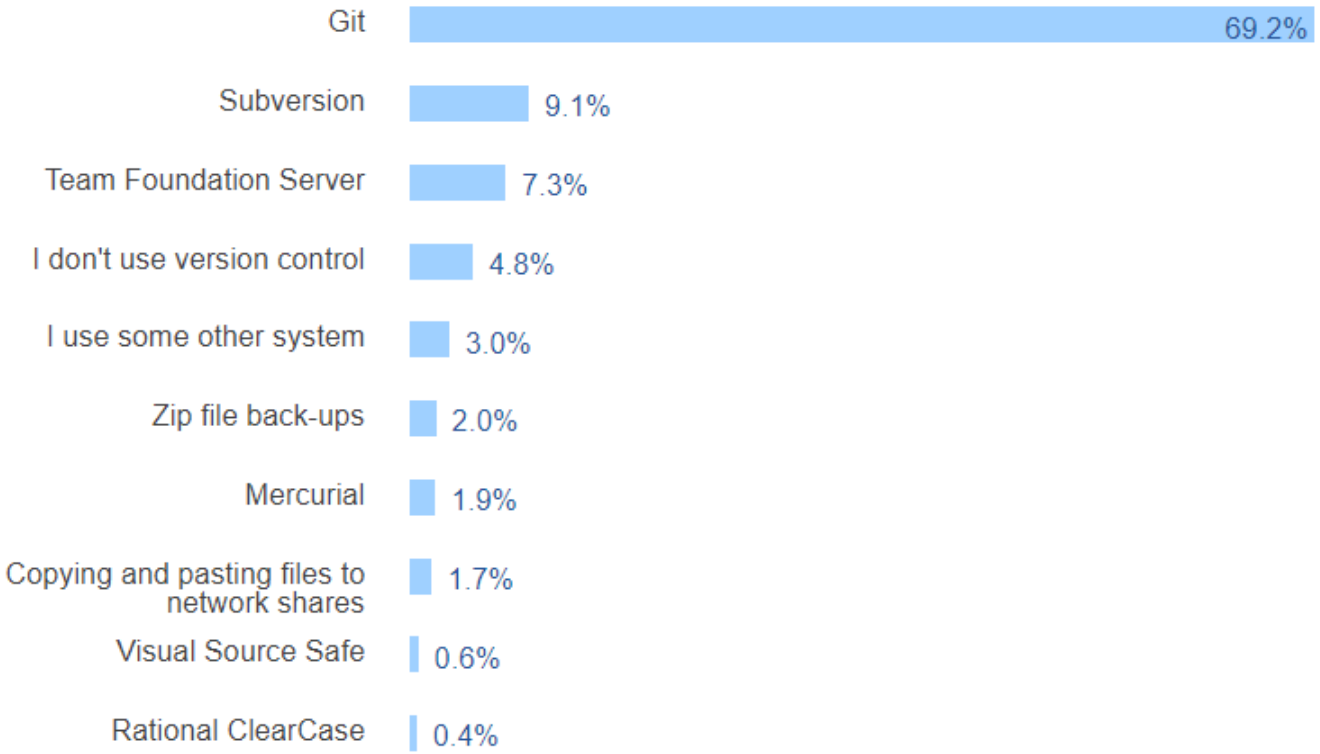
When using a version control system, you have **one copy of each file** and the *version control system tracks the changes* that have occurred over time

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
69.2%
Subversion
9.1%
Team Foundation Server
7.3%
I don't use version control
4.8%
I use some other system
3.0%
Zip file back-ups
2.0%
Mercurial
1.9%
Copying and pasting files to network shares
1.7%
Visual Source Safe
0.6%
Rational ClearCase
0.4%

git & GitHub

“Global Information Tracker”

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.

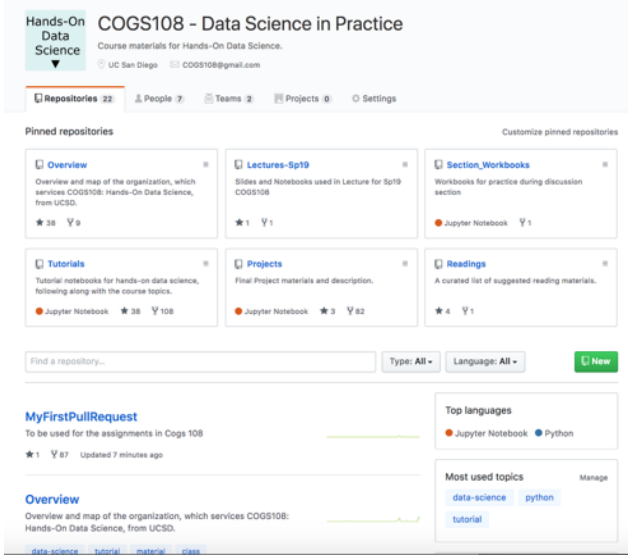
~ Dropbox….but way better

What version control looks like

```
$ git clone https://www.github.com/username/repo.git
$ git pull
$ git add -A
$ git commit -m “informative commit message”
$ git push
```

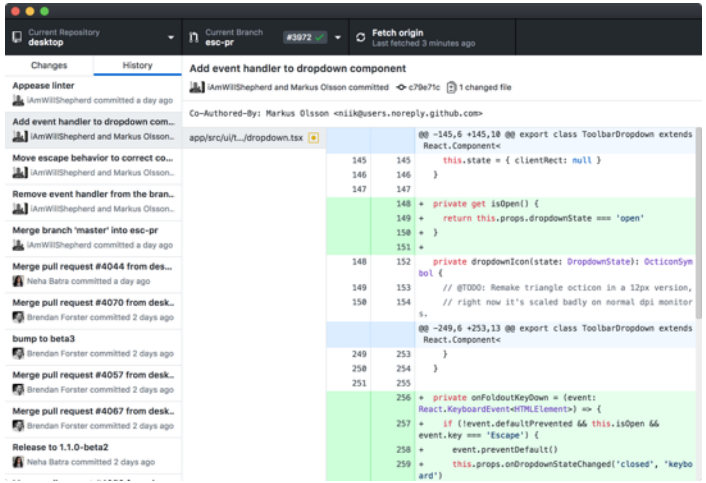
Terminal

git

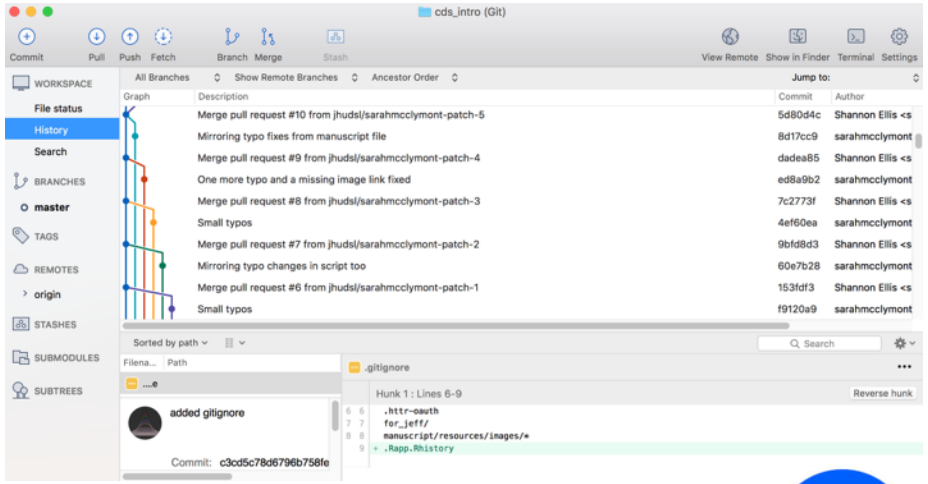


GitHub

GUIs can be helpful when working with version control



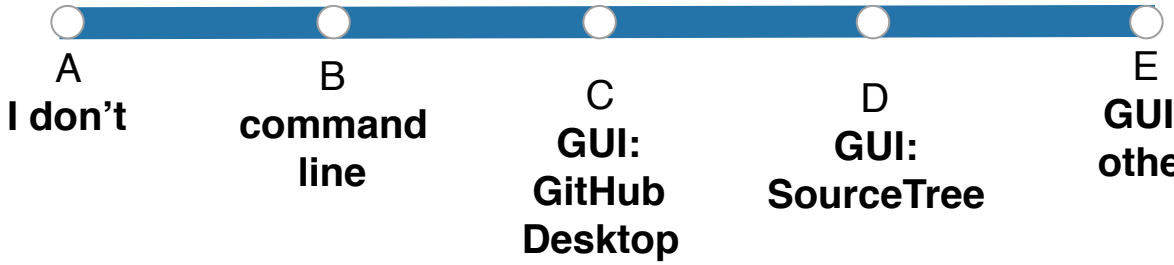
GitHub Desktop



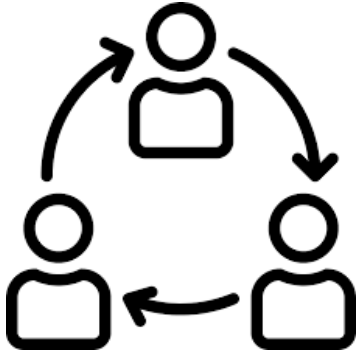
SourceTree

Version Controller

How do you typically interact with git?



Why version control with git and GitHub?



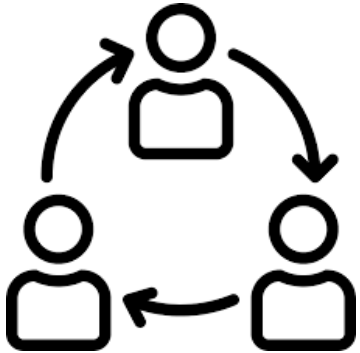
Collaboration

Returning to a safe state

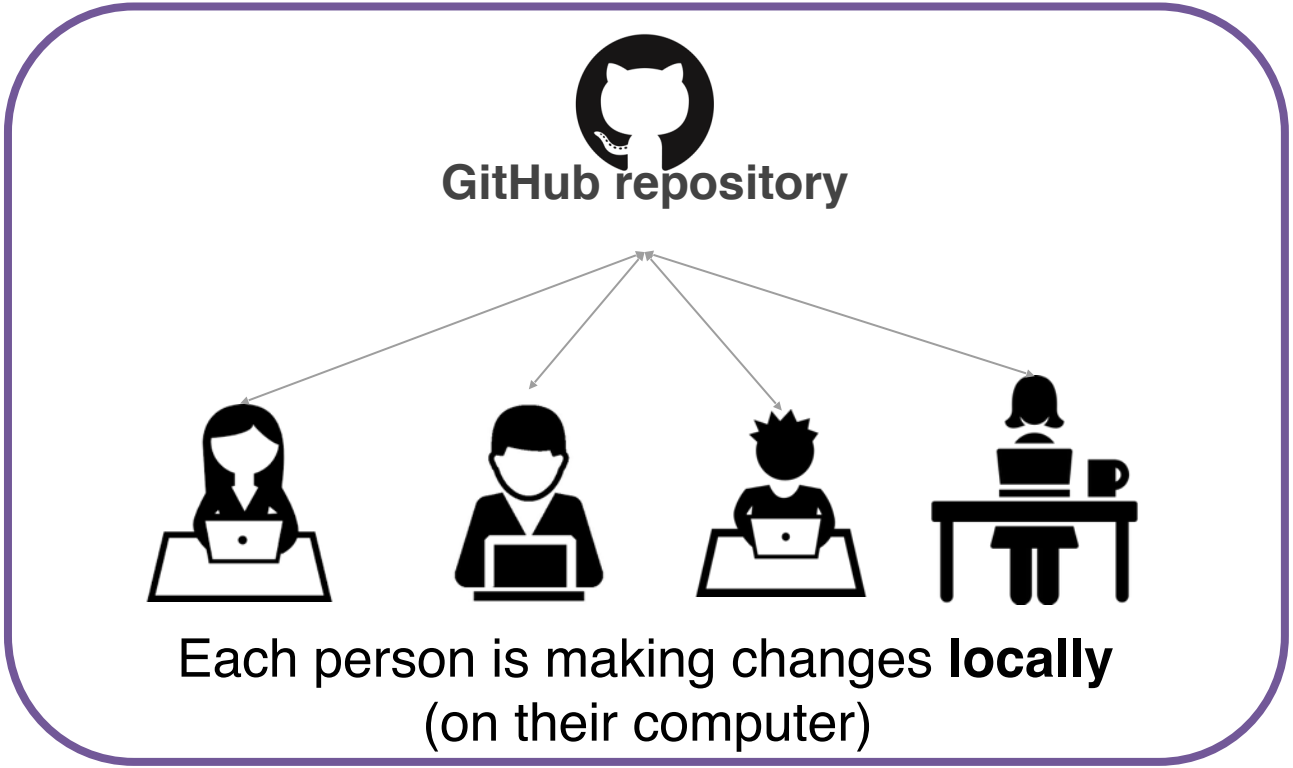
Exposure for your work

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!

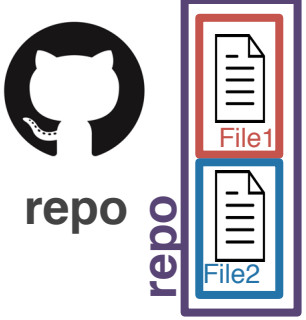
Exposure for your work

Your public GitHub repos are your coding social media

Keep up with others' work easily

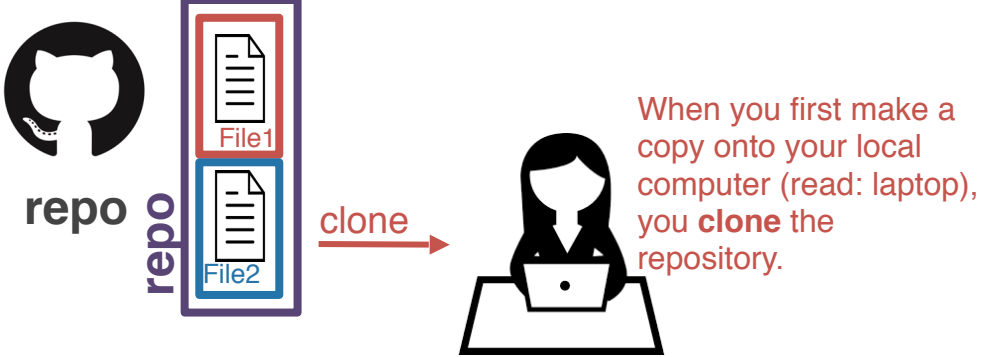
Tracking others' work

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

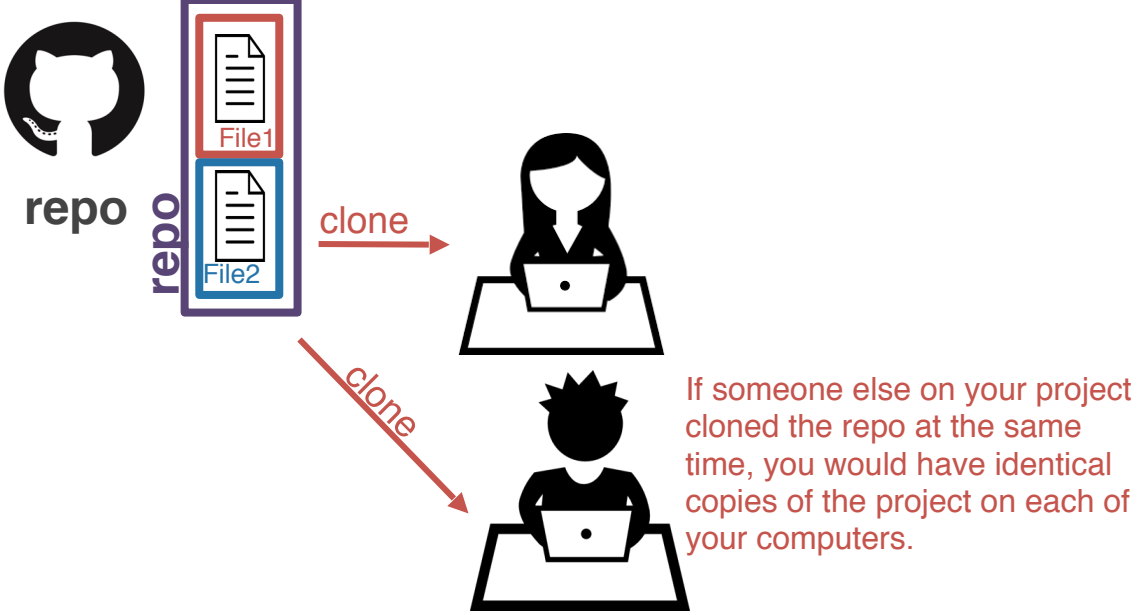


A **GitHub repo** contains all the files and folders for your project.

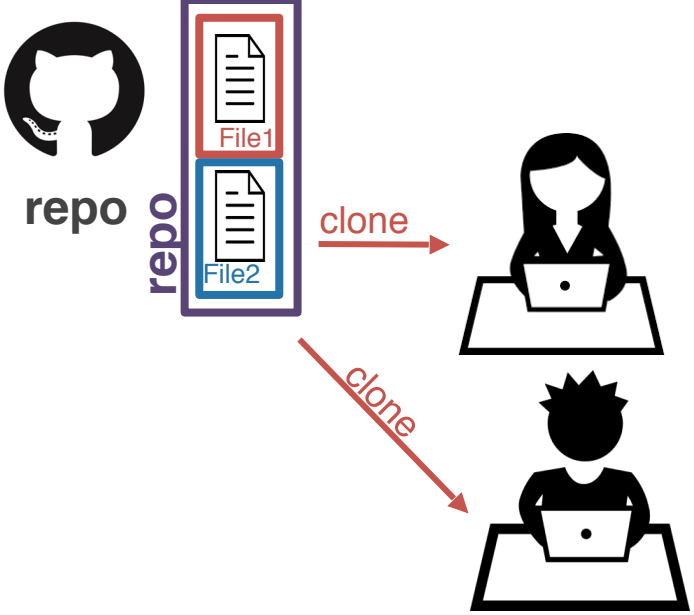
GitHub is a **remote host**. The files are geographically distant from any files on your computer.



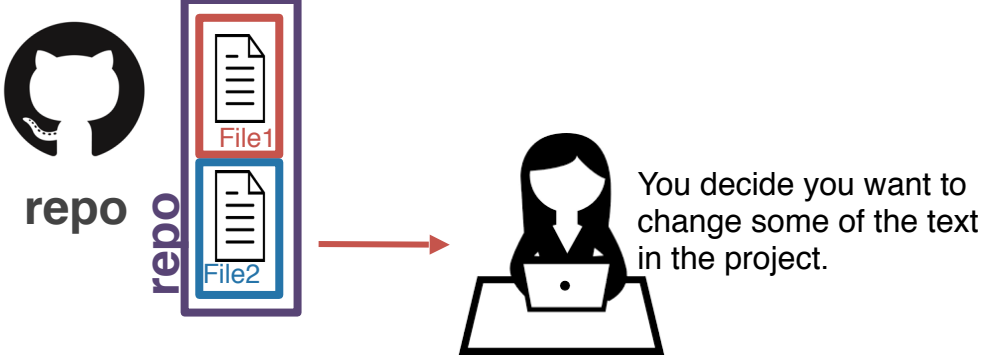
When you first make a copy onto your local computer (read: laptop), you **clone** the repository.

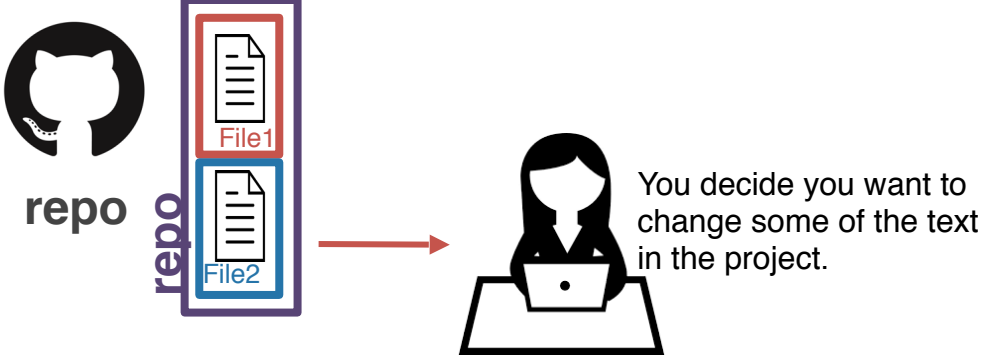


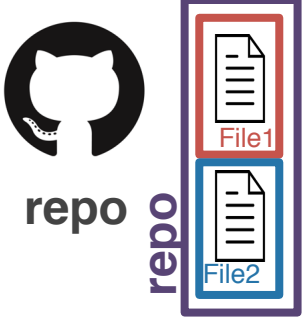
If someone else on your project cloned the repo at the same time, you would have identical copies of the project on each of your computers.

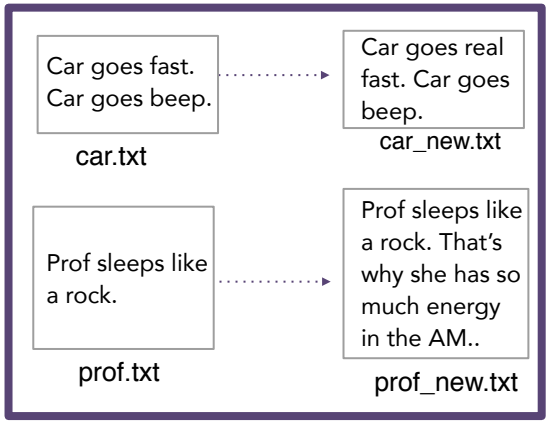


Yay! Everyone can work on the project!

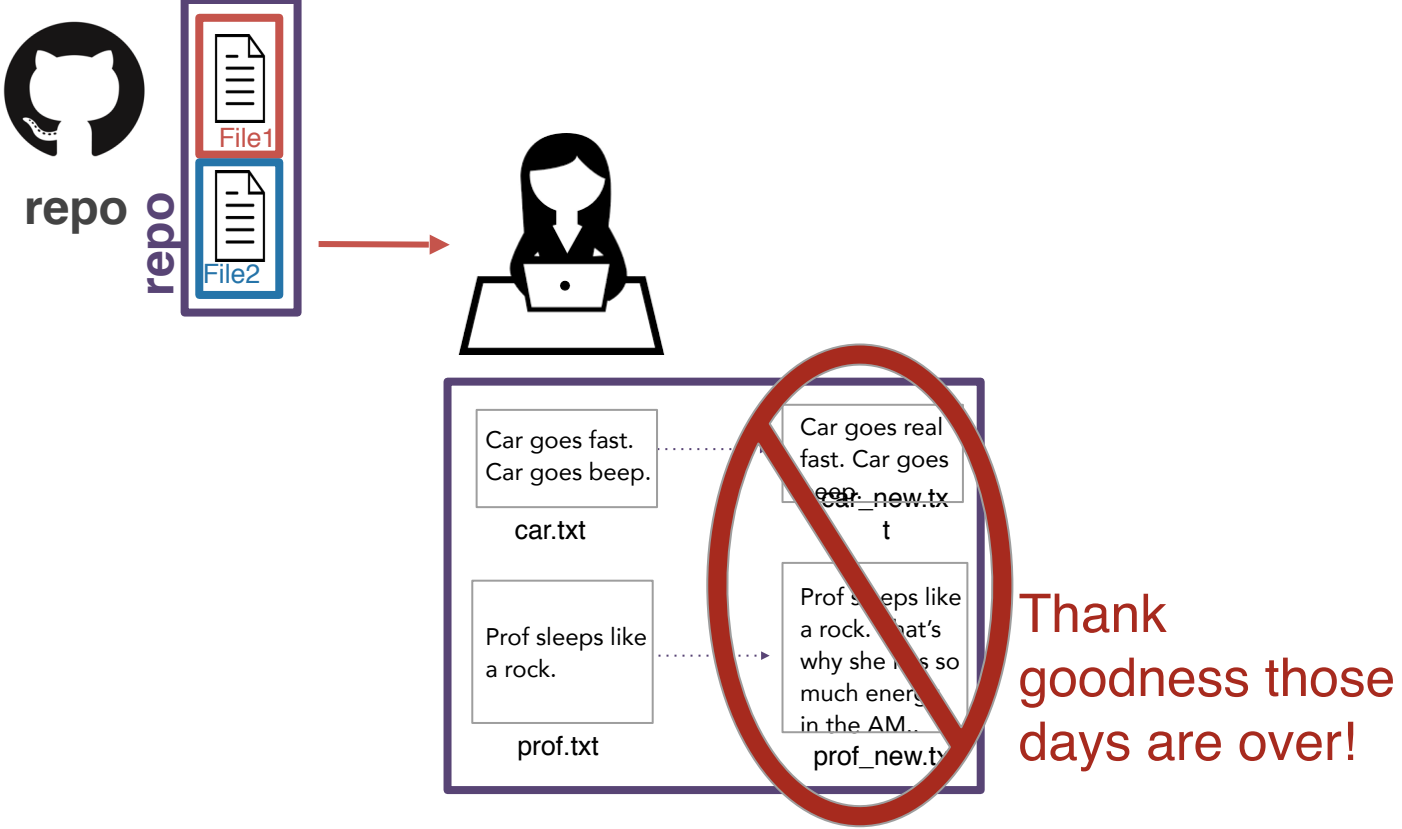

repo
repo
File1
File2
You decide you want to change some of the text in the project.

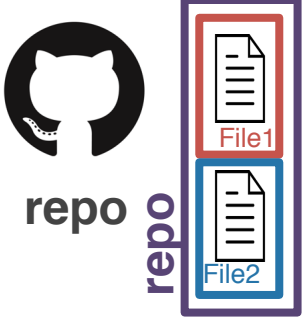

repo
repo
File1
File2
You decide you want to change some of the text in the project.

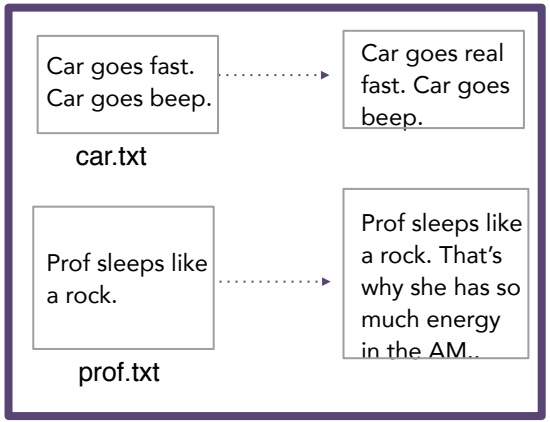




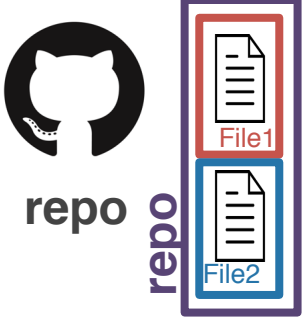
without git...you'd likely rename these files….

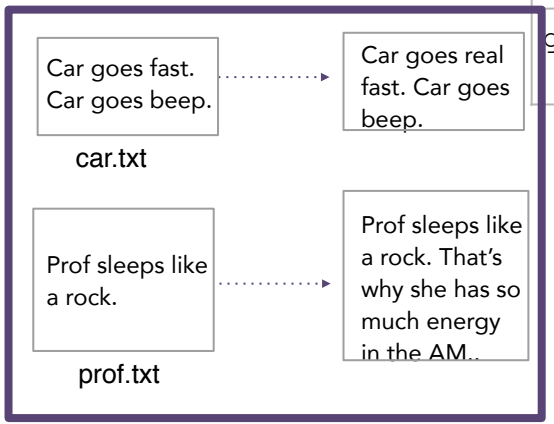

repo
repo
File1
File2
Car goes fast.
Car goes beep.
car.txt
Car goes real
fast. Car goes
t
Prof sleeps like
a rock.
prof.txt
Thank
goodness those
days are over!





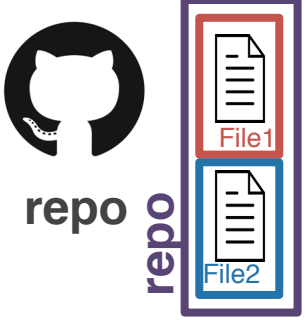
Instead, you tell git which files you'd like to keep track of using **add**. This process is called *staging*.

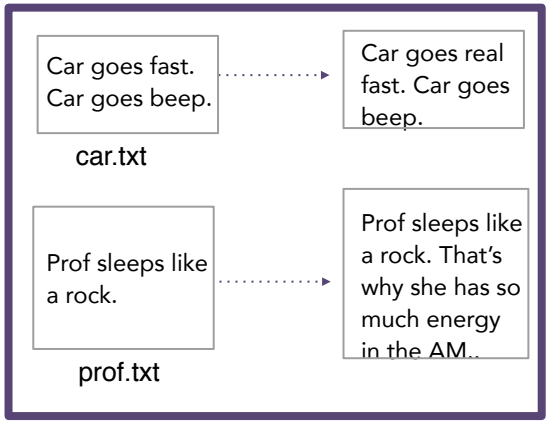




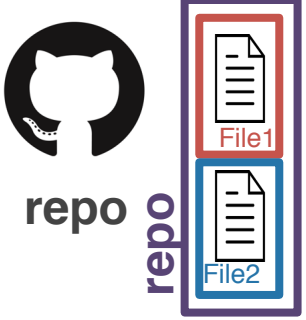
`git add file`	stages specified file (or folder)
`git add .`	stages new and modified files
`git add -u`	stages modified and deleted files
`git add -A`	stages new, modified, and deleted files
`git add *.csv`	Stages any files with .csv extension
`git add *`	Use with caution: stages everything

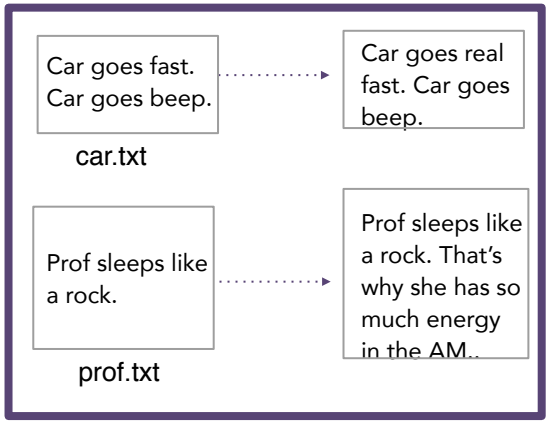
Instead, you tell git which files you'd like to keep track of using **add**. This process is called *staging*.





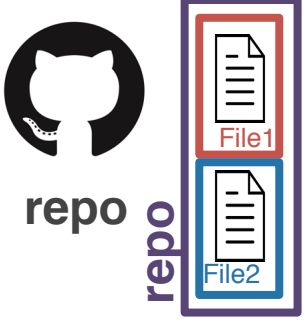
Then, you create a snapshot of your files at this point. This snapshot is called a **commit**.

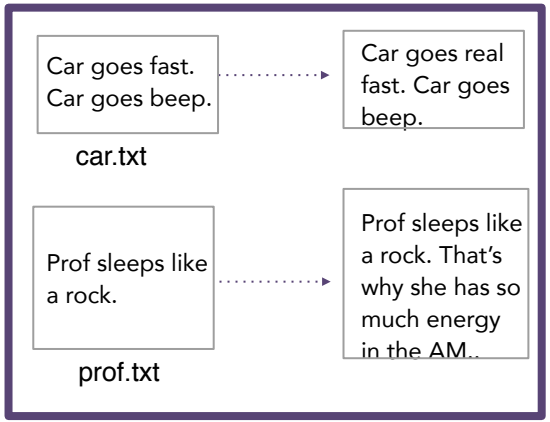




Then, you create a snapshot of your files at this point. This snapshot is called a **commit**.

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



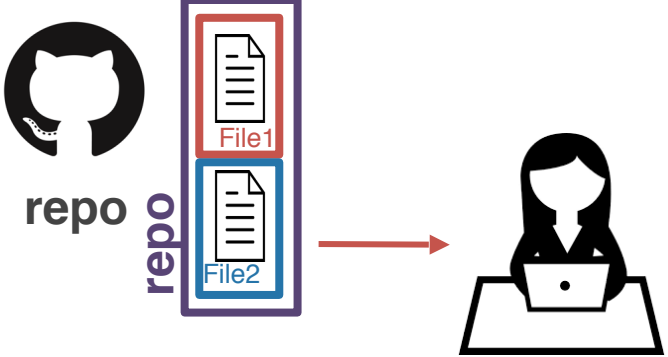


You can make commits more informative by adding a **commit message**.

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

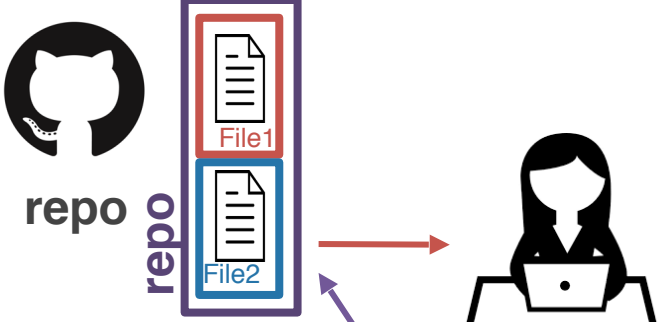
Then, you create a snapshot of your files at this point. This snapshot is called a **commit**.

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



Shannon Ellis
3/28/21 3:28pm

fix typos in car and prof

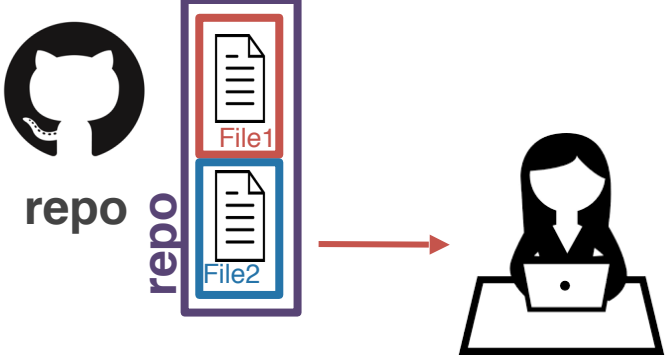


push

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



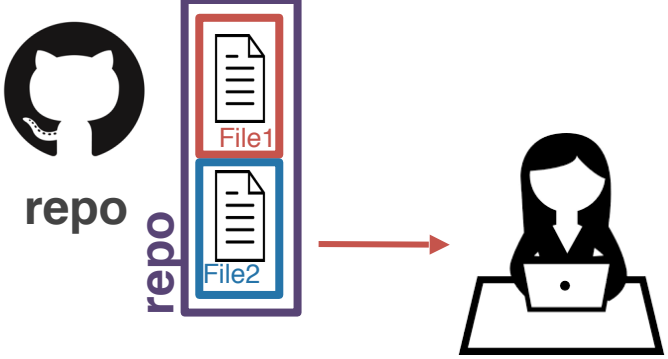
Shannon Ellis
3/28/21 3:28pm

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



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

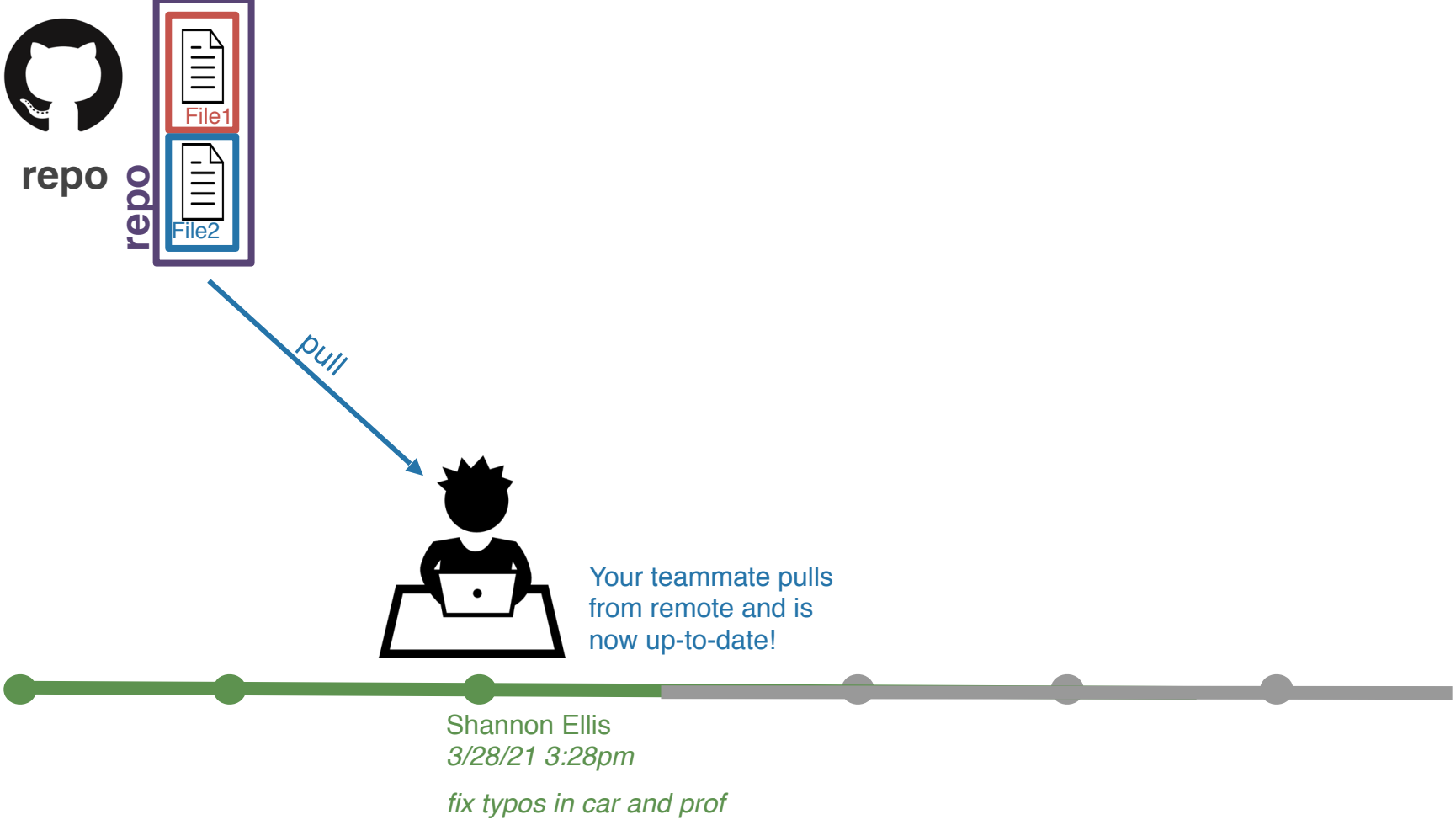
Shannon Ellis
3/28/21 3:28pm

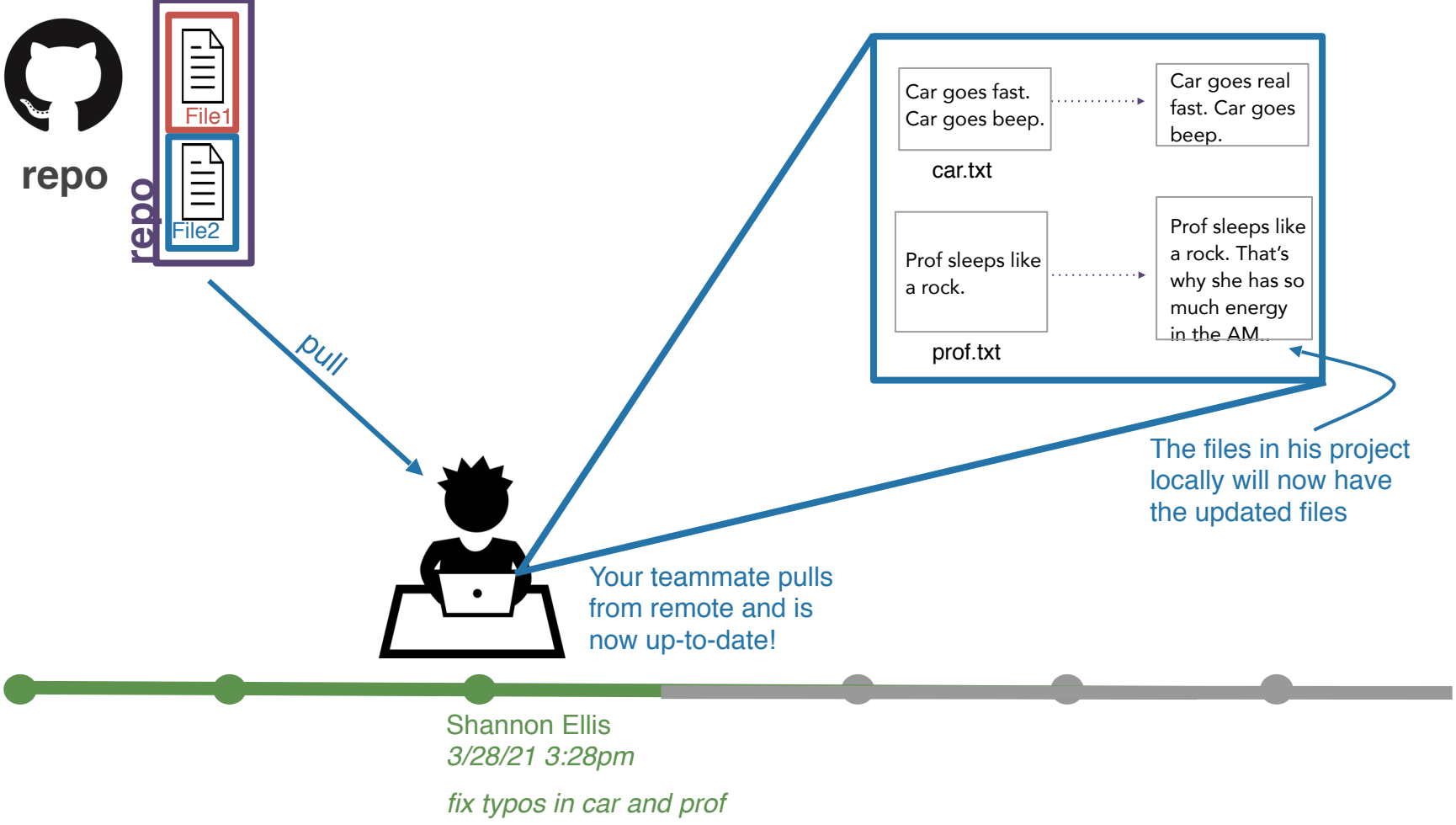
fix typos in car and prof

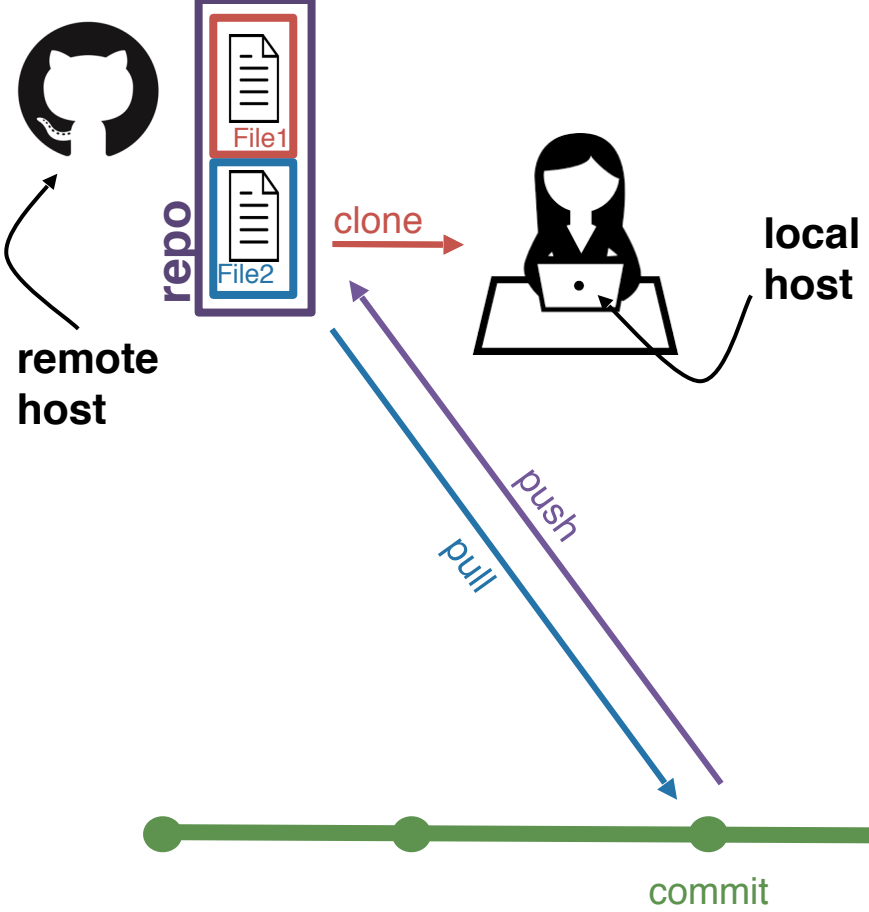
Your teammate is still working with the (out-of-date) copy he cloned earlier!

Shannon Ellis
3/28/21 3:28pm

fix typos in car and prof


repo
repo
File1
File2
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
repo
File1
File2
pull
Car goes fast. Car goes beep.
Car goes real fast. Car goes beep.
car.txt
Prof sleeps like a rock.
Prof sleeps like a rock. That's why she has so much energy in the AM.
prof.txt
The files in his project locally will now have the updated files
Your teammate pulls from remote and is now up-to-date!
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

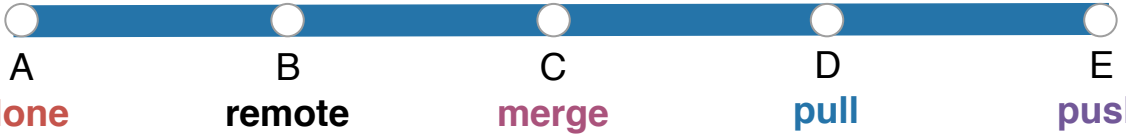
```
(base) sellis:Projects shannonellis$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        FinalProject_Guidelines.pdf

nothing added to commit but untracked files present (use "git add" to track)
(base) sellis:Projects shannonellis$ git add FinalProject_Guidelines.pdf
(base) sellis:Projects shannonellis$ git commit -m "update Project Guidelines"
[master 264e91a] update Project Guidelines
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 FinalProject_Guidelines.pdf
(base) sellis:Projects shannonellis$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 148.21 KiB | 29.64 MiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/COGS108/Projects.git
   6931768..264e91a  master -> master
```

Review & Question Time

Version Controller I

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.

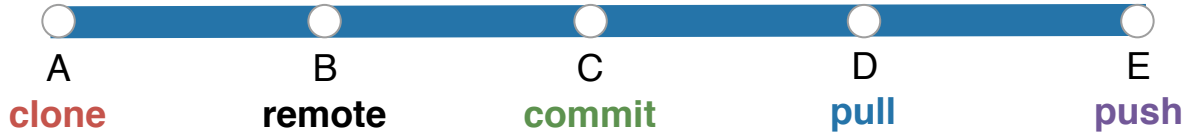
What do you do to get them on the remote?

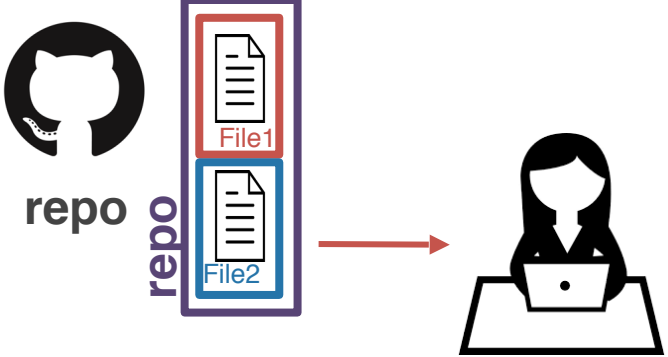


Version Controller II

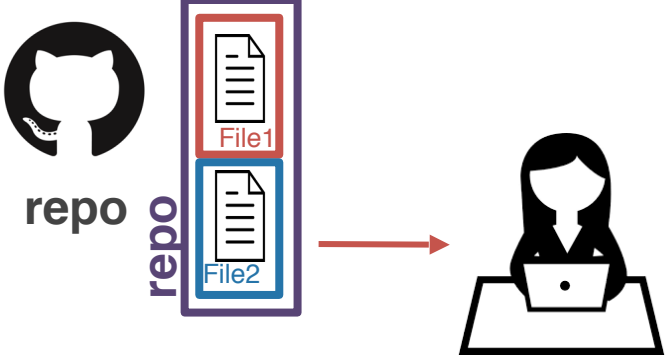
Your teammate has given you access to a GitHub repository to work on a project together. You want to get them for the first time on your computer locally.

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

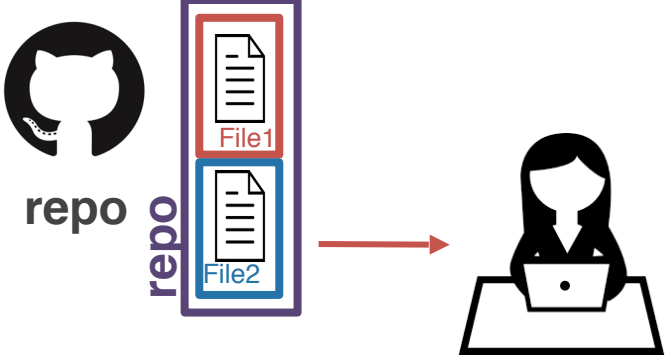




Each time you create a commit, git tracks the changes made automatically.



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



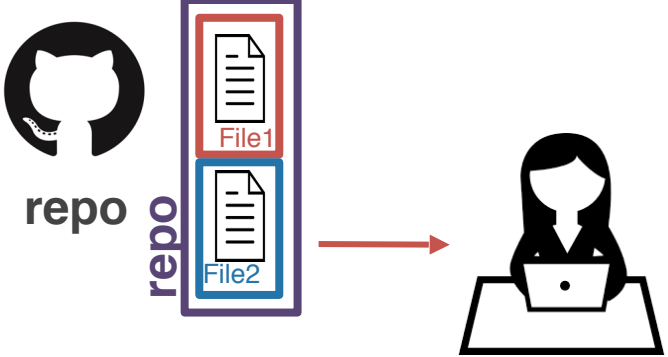
By committing each time you make changes, git allows you to time travel!

377dfcd00dd057542b112cf13be6cf1380b292ad

439301fe69e8f875c049ad0718386516b4878e22

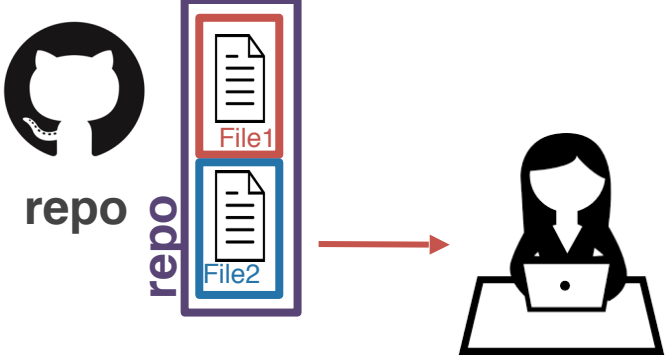
456722223e9f9e0ee0a92917ba80163028d89251

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

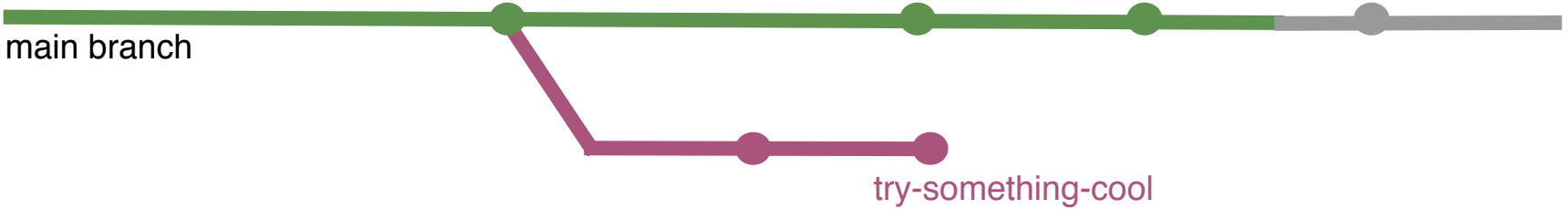


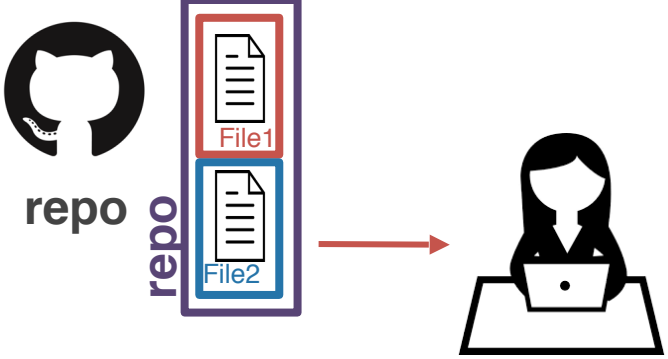
You can return to the state of the repository at any commit. Future commits don't disappear. They just aren't visible when you **check out** an older commit.

377dfcd00dd057542b112cf13be6cf1380b292ad

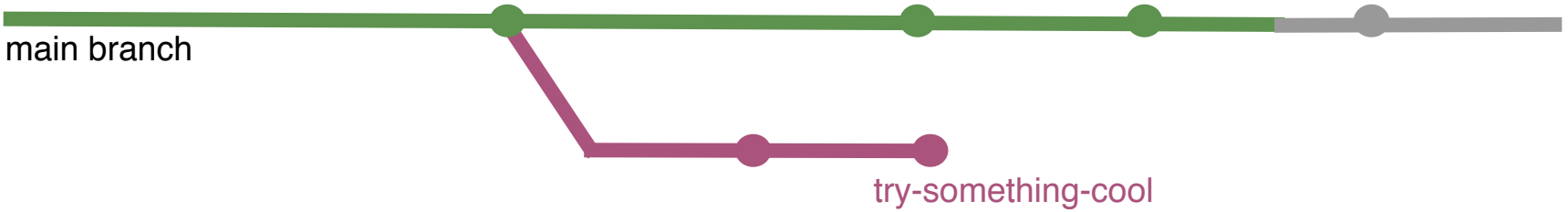


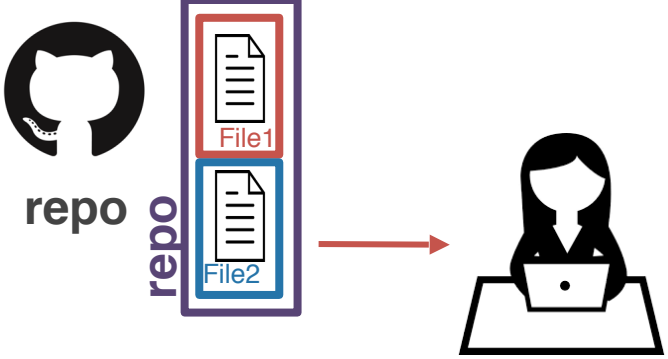
But...not everything is always linear. Sometimes you want to try something out and you're not sure it's going to work. This is where you'll want to use a **branch**.



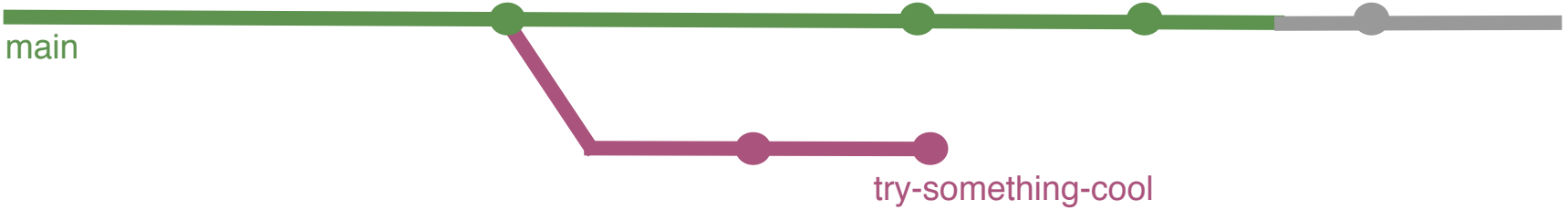


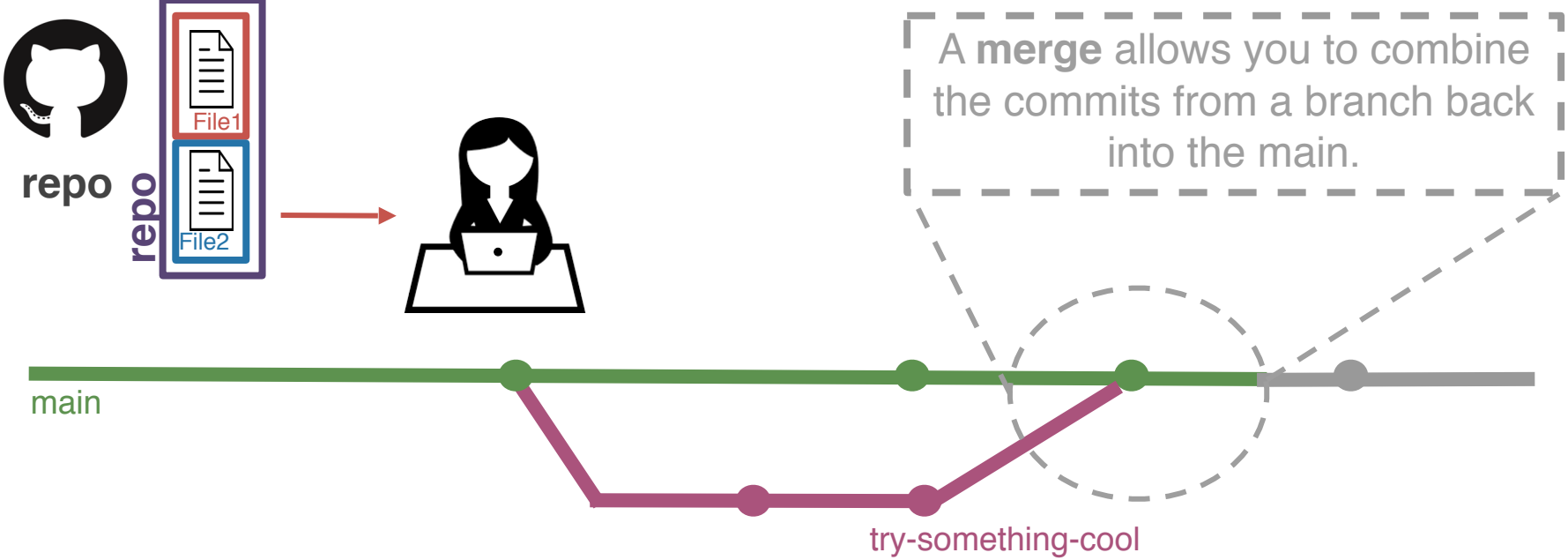
It's a good way to experiment. It's pretty easy to get rid of a branch later on should you not want to include the commits on that branch.

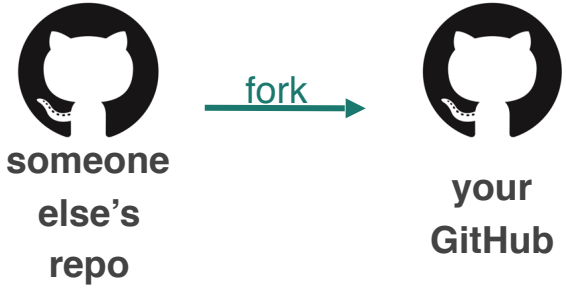




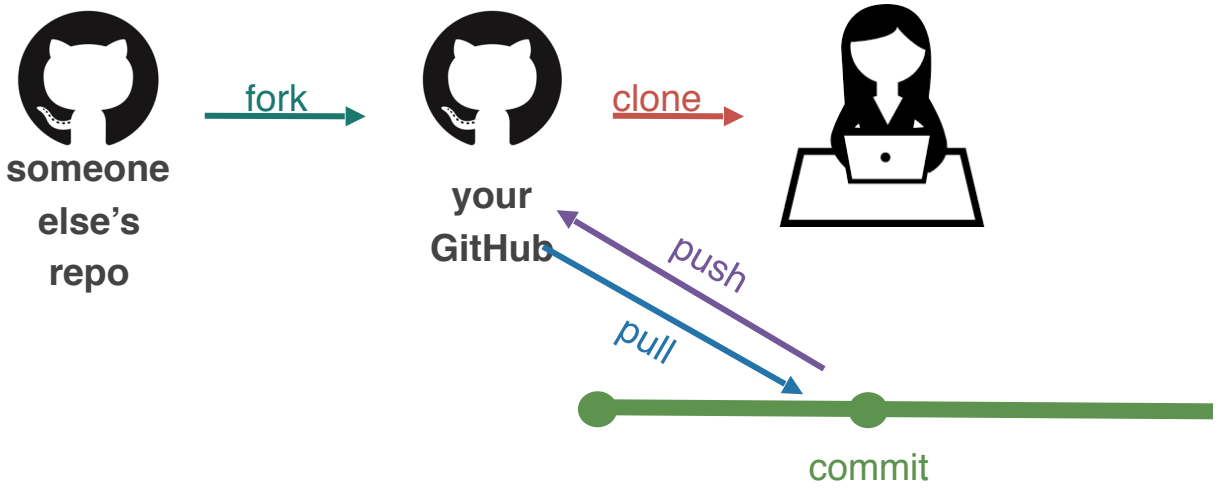
But...what if you DO want to include the changes you've made on your try-something-cool branch into the main branch?



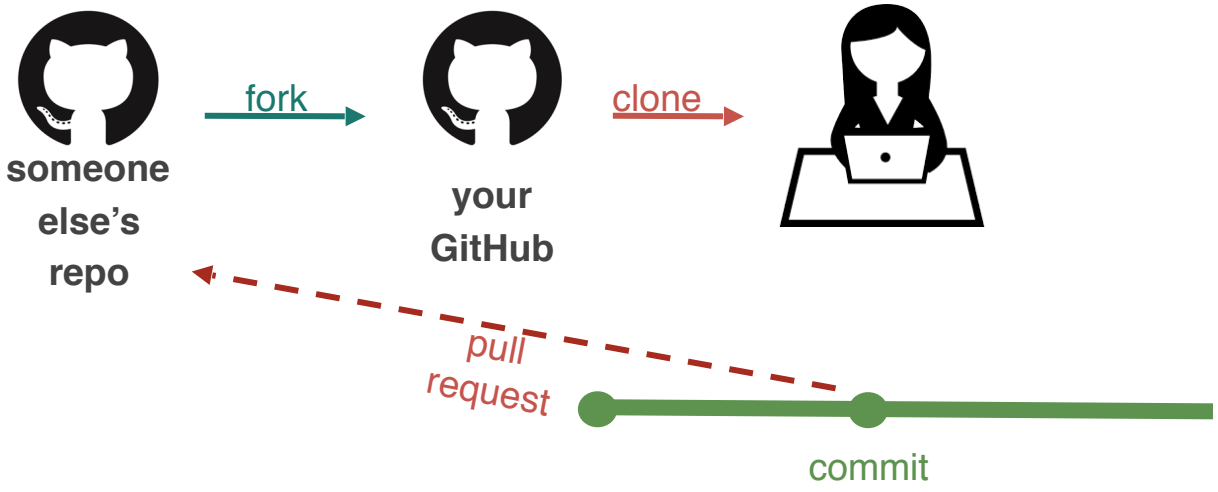

repo
repo
File1
File2
A **merge** allows you to combine the commits from a branch back into the main.
main
try-something-cool



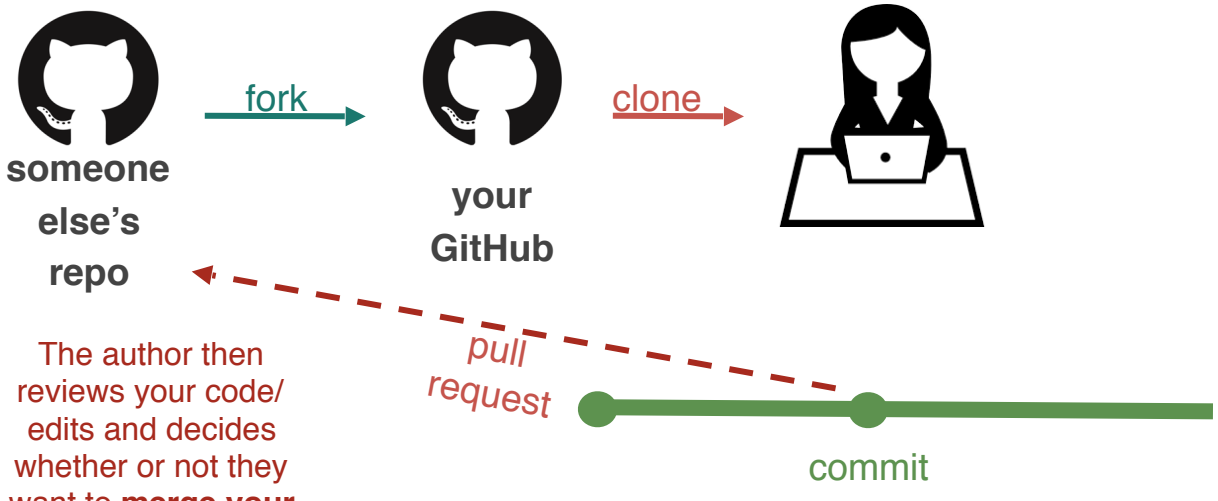
What if someone else is working on something cool and you want to play around with it? You'll have to **fork** their repo.



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**).



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.

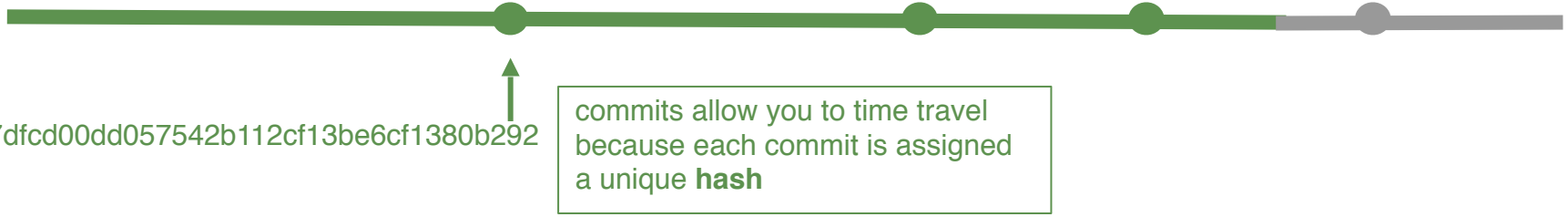
someone else's repo

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!

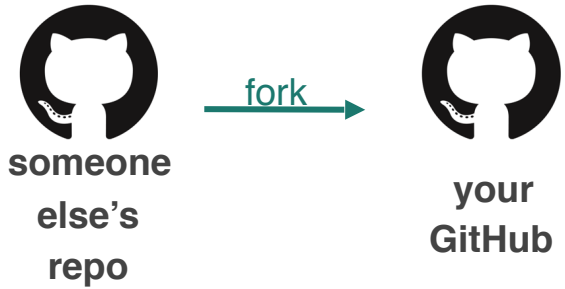
They can be assigned to people.

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



One more git recap...

One more git recap...



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

One more git recap...

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

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

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

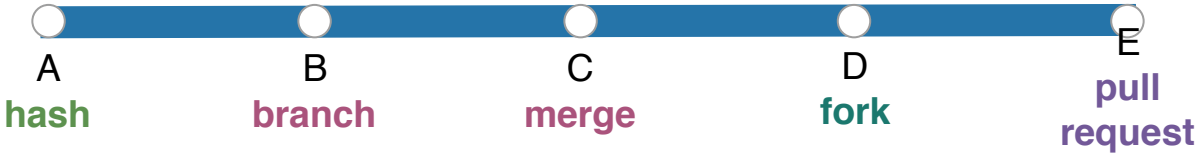
One more git recap...

Review & Question Time

Version Controller III

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

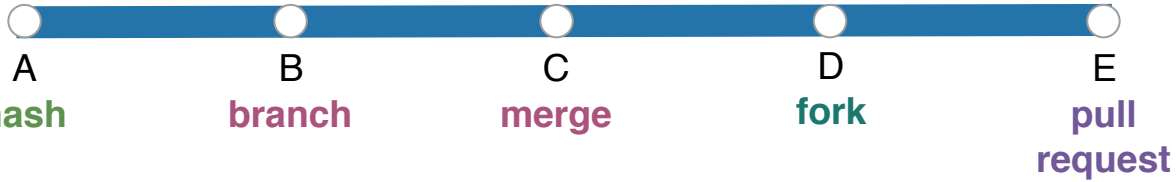
what should you do?



Version Controller IV

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?



Jupyter notebooks have problems for version control

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

Version Control: Practice

- Discussion Lab 1: Part 3
- Assignment 1: Part 1
  - This will get you practice with git & GitHub
  - Understand what you're doing in the assignment!
  - You may have to google, ask others, spend some time with this!
  - Part II is a Python review; each part of this assignment is self-contained
  - Do this part of the assignment ASAP
- 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

Note: You're encouraged to put projects on GitHub. Please do not put assignments on GitHub.

COGS 108 Final Projects

The COGS 108 Final Project will give you the chance to explore a topic of your choice and to expand your analytical skills. By working with real data of your choosing you can examine questions of particular interest to you.

- You are encouraged to work on a topic that matters to the world (your family, your neighborhood, a state/province, country, etc).
- Taboo Topics: Movie Predictions/Recommendation System; YouTube Data Analysis, Kickstarter success prediction/analysis, prediction of what makes a song popular on Spotify; political patterns or singling out some individual, anything that violates university policy

Final Project: Objectives

- Identify the problems and goals of a *real* situation and dataset.
- Choose an appropriate approach for formalizing and testing the problems and goals, and be able to articulate the reasoning for that selection.
- Implement your analysis choices on the dataset(s).
- Interpret the results of the analyses.
- Contextualize those results within a greater scientific and social context, acknowledging and addressing any potential issues related to privacy and ethics.
- Work effectively to manage a project as part of a team.

Upcoming Project Components

Project Review (5%) - Before Mon of week 2, your group will be assigned a previous COGS 108 project to review; A google Form will be released to guide your thinking/discussion about and review of what a previous COGS 108 group did for their project. (due Fri Week 2)

Project Proposal (9%) - a GitHub repo will be created for your group; ‘submit’ on GitHub (due Fri Week 2)

Project Proposal (9%)

Full project guidelines are going to be at the projects link of the web page: http://casimpkinsjr.radiantdolphinpress.com/pages/cogs108_ss1_23/index.html