

TERMINAL, GIT, AND GITHUB





IAP 15.S60 Session 1
Alex Schmid

Today's Learning Objectives

- Describe the importance of reproducibility in coding projects
- Enter basic commands in terminal to navigate a file system, manipulate files, and run scripts
- Maintain proper version control for a coding project using git
- Collaborate on a coding project with Github

Reproducibility

Motivation for today's class
Adapted from slides by Sam Gilmour and the Turing Way

What is reproducibility?

"The ability to obtain consistent results no matter the machine a project runs on."

		Data		
		Same	Different	
Analysis	Same	Reproducible	Replicable	
	Different	Robust	Generalisable	

Tools for reproducibility

- Version control maintain a history of your project to better track changes and decisions
- Testing simple tests to detect errors and changes in functionality when new features are added
- Maintaining a reproducible computational environment for example, software and package versions
- Making code and data open and available when possible

My reproducibility nightmare

- In my first PhD research project, I created a new optimization model and algorithm for a problem. I ran my computational experiments without reproducibility in mind.
- Months later, we finished writing the paper and stumbled on a new model that was wayyyyy better and faster. Welp... Time to re-run everything!
 - Spent a day trying to get code set up to run again
 - New results didn't match up with original results
 - Eventually, I tracked down a single parameter I had changed
 - I changed the parameter and re-ran everything again
 - Took me weeks of work and huge amounts of cluster resources

Think-Pair-Share Activity

• Think about a non-reproducible or non-auditable workflow you have used before at work, in research, on a personal project, or in coursework that negatively impacted your work somehow (2 minutes)

Pair with the person beside you and share your story and how it negatively impacted you (3-4 minutes)

We'll come back together as a class and share a few stories

What is reproducibility good for?



Tools for reproducibility

- Version control maintain a history of your project to better track changes and decisions
- Testing simple tests to detect errors and changes in functionality when new features are added
- Maintaining a reproducible computational environment for example, software and package versions
- Making code and data open and available when possible

Another Think-Pair-Share

 Think about barriers that prevent you (or may prevent others) from using a reproducible and auditable workflow. What makes reproducibility a hard standard to meet? (2 minutes)

 Pair with the person beside you and share 1-2 things that might make reproducibility hard for you or others (2 minutes)

We'll come back together as a class and share some ideas

Failure Case

Clinical trials for 110 cancer patients hoping to identify personalized treatments based on gene signatures



Strong results, highimpact publications



Experts find errors in analysis

- Excel formula errors
- Poor documentation



Trials terminated,
25 papers retracted

Tools for Reproducible Research

Git - Version control system



- Command line tool
- Used to make and document local code changes (and communicate those changes with Github)

Github - Hosting platform for version control and collaboration



- Widely used by developers to store their projects
- Easily share code and data, privately and publicly

Terminal

Adapted from slides by Galit Lukin, Jackie Baek

What is the terminal?

The terminal/Unix shell is a text-based interface to interact with the computer.

```
MINGW64:/c/Users/13362/15.S60 2022
                                                                                 X
L3362@DESKTOP-QIIOQGP MINGW64 ~ (main)
 hey terminal
bash: hey: command not found
L3362@DESKTOP-QIIOQGP MINGW64 ~ (main)
$ cd 15.s60_2022
L3362@DESKTOP-QIIOQGP MINGW64 ~/15.S60_2022 (main)
```

Why use the terminal?



Repetitive tasks, like

"delete all files in a

directory ending in .csv"





Chain commands across programming languages sequentially





Access to client servers and computing clusters with Secure Shell (SSH)

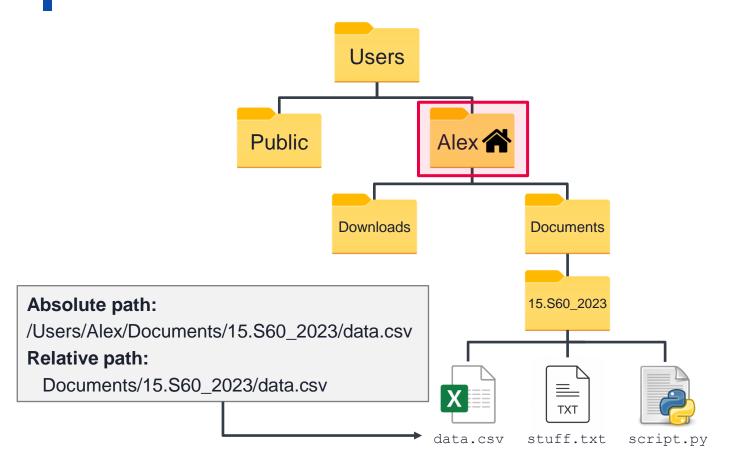
We must learn a few basic commands to interact!

Learning Objectives - Terminal

At the end of the session, students will be able to...

- Navigate directories and manipulate files using terminal commands
- Recognize scenarios when it is necessary or more efficient use the terminal rather than a graphical interface

Files and Directories



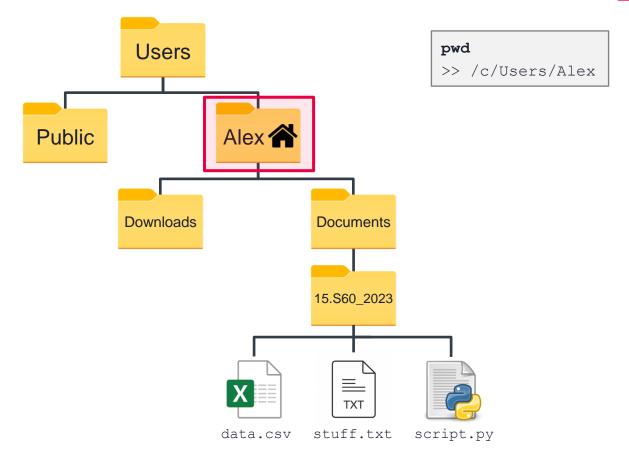
Terminal Basics

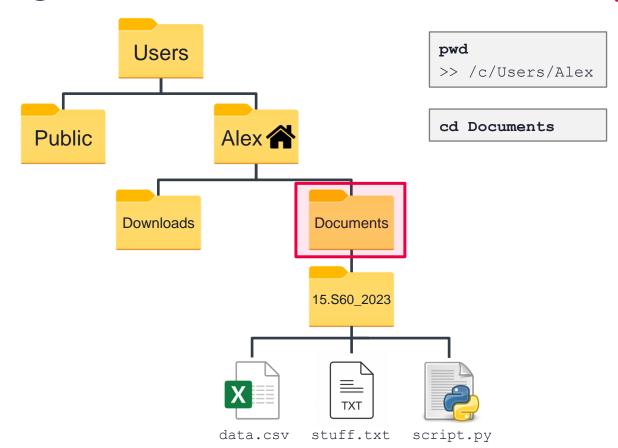
- We are using a shell called bash. This program will interpret and process the commands you input into the terminal.
- A typical command looks like:

```
command <argument1> <argument2> ...
```

To open:

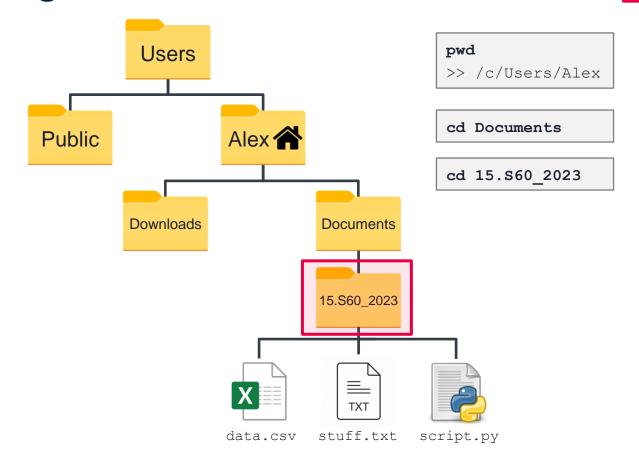
- Mac users open Terminal
- Windows users open Git Bash (installed in the pre-assignment)





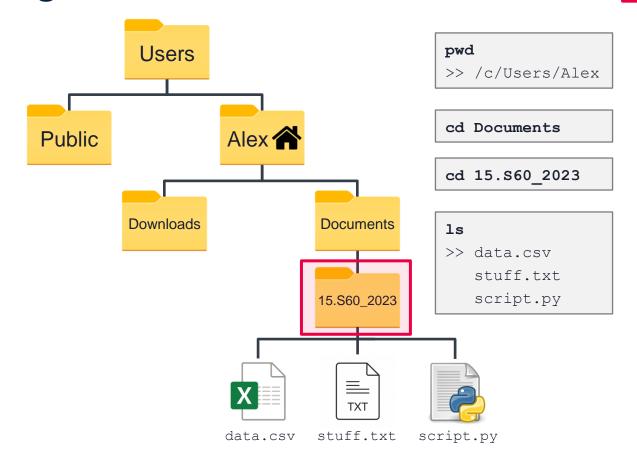
Working directory

Navigating – Commands

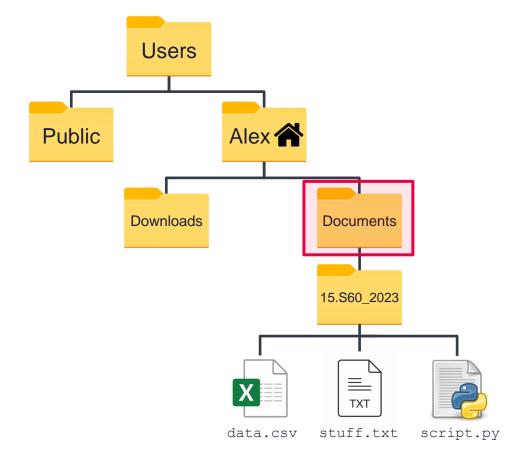


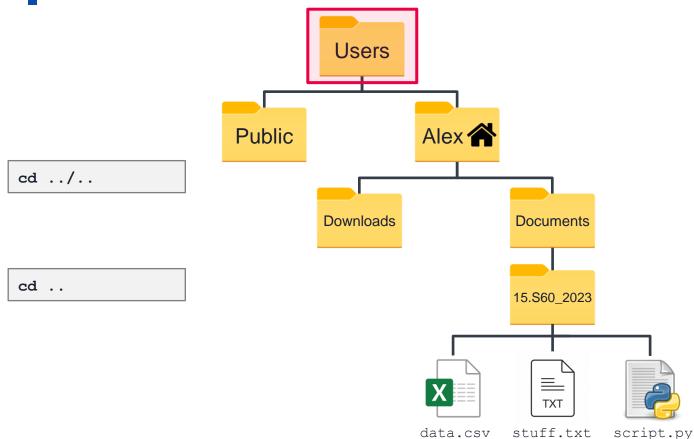
Working directory

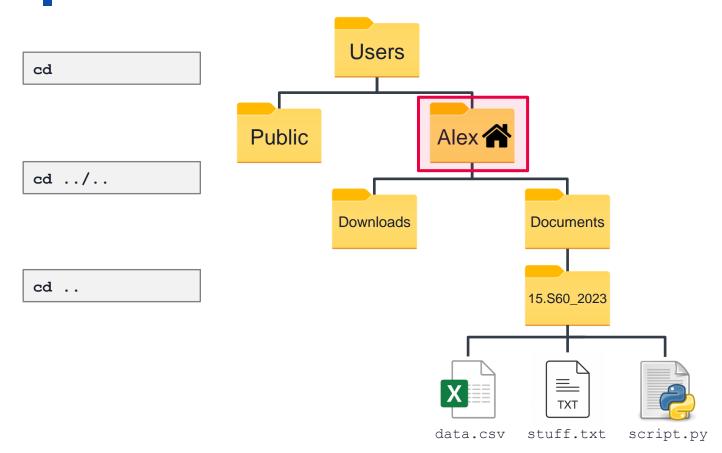
Navigating – Commands



cd ..







Print working directory	pwd
List contents of working directory	ls
List contents of specified directory	ls <directory_name></directory_name>
Change to a new directory	cd <directory_name></directory_name>
Return to "home" directory	cd
Open a file (analogous to double clicking)	open <filename></filename>

Reference current working directory	•
Reference parent of current working directory	• •
Reference home directory	~

- Navigate history of previous commands using up and down arrow keys
- Use tab to autocomplete commands and paths

Poll Question

pollev.com/alexandriaschmid171

If our current working directory is <code>Users/Alex/Documents</code>, which commands would take us back to the home directory, <code>Users/Alex?</code> (select all that apply)

- A.) cd ..
- B.) cd ../..
- **C.)** cd ~
- **D.)** cd .
- E.) cd

Navigation – Try it out

- 1. Open Terminal (MacOS) or Git Bash (Windows)
- 2. Navigate to the class directory from the pre-assignment

3. Pull the class Github to get any updates

4. Navigate to the class directory from the pre-assignment

- 5. Print off the list of files in that directory
- 6. Navigate back to your home directory

File Manipulation – Commands

Move or rename a file

Create a new directory	mkdir <directory_name></directory_name>
Create a new file	touch <filename></filename>
Delete a file (cannot be undone!)	rm <filename></filename>
Edit file contents (Nano is a text editor)	nano <filename></filename>
Print contents of a file	cat <filename></filename>

mv <source filename> <target filename>

File Manipulation – Demo

We want to create a new directory called mydirectory, navigate to it, add a new file called myfile.txt, then add a line of text to the file.

1. Create and navigate to a new directory

```
mkdir mydirectory
  cd mydirectory
```

2. Add and open a new file for editing

```
touch myfile.txt
nano myfile.txt
```

3. Add some text, save (Ctrl+O), and exit Nano (Ctrl+X)

File Manipulation – Try it out

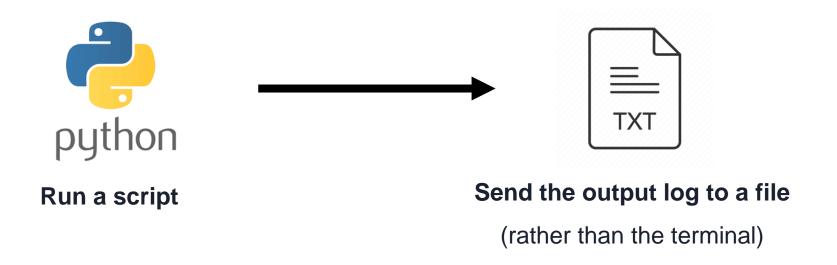
We want to create a directory within mydirectory called data, navigate to it, add a new file called mydata.csv. Then navigate back to mydirectory.

1. Within mydirectory, create and navigate to a new directory

2. Add a new CSV file

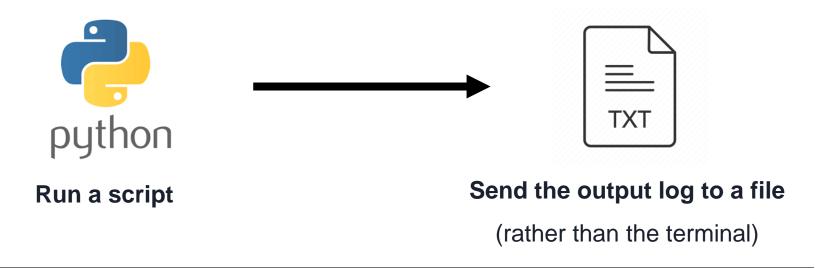
3. Navigate back to mydirectory

Redirecting Outputs



python processStuff.py > outputfile.txt

Redirecting Outputs



Overwrite / create file Append to file <command> > outputfile.txt

<command> >> outputfile.txt

Redirecting Outputs - Demo

I can create a Julia script in mydirectory called myscript.jl. I can then run it and direct the output to a file called scriptoutput.txt.

1. Create new text file

2. Print the contents of the directory and redirect the output

```
julia myscript.jl > scriptoutput.txt
```

3. Check the output:

```
cat scriptoutput.txt
```

Redirecting Outputs - Try it out

Create another text file in mydirectory called newfile.txt. Print the contents
mydirectory and direct the list to a file called outputfile.txt.

What is Bash actually doing?

Two types of commands, internal (built-in to the shell) and external

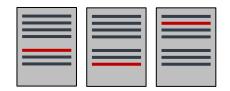
```
13362@DESKTOP-QIIOQGP MINGW64 ~/myrepo (master)
$ type cd
cd is a shell builtin

13362@DESKTOP-QIIOQGP MINGW64 ~/myrepo (master)
$ type cat
cat is hashed (/usr/bin/cat)
```

When you run an external command like cat, bash looks for a program with that name under the directories listed in the \$PATH environment variable and spawns a process that executes the program

More Terminal

There are many more things you can do in Terminal.



Simple pattern matching to find and sort files



Use multiple programming languages to run sequential processes



Shell scripts to automate chains of commands you use often

If you're interested, check out the tutorial here: https://swcarpentry.github.io/shell-novice/

Break

Resume in 10 minutes



Adapted from slides by Galit Lukin and Jackie Baek with activities from Turing Way and Software Carpentry

Learning Objectives – Git/Github

At the end of the session, students will be able to:

- Maintain a version control history with appropriate documentation
- Contribute to a project with shared code
- Share and contribute code to the broader community

Git and Github

Github - Hosting platform for version control and collaboration



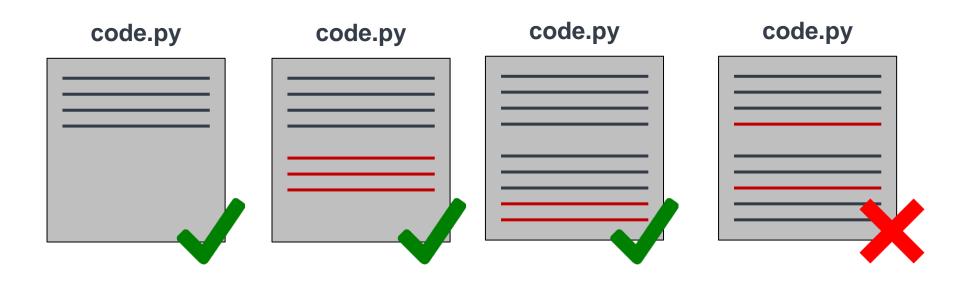
- Widely used by developers to store their projects
- Easily share code and data, privately and publicly

Git - The version control system itself



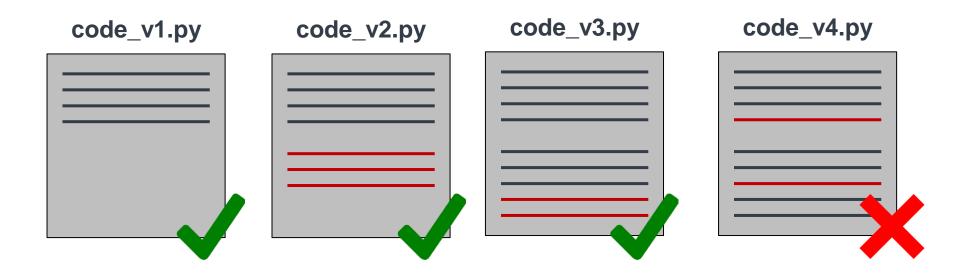
- Command line tool
- Used to make local code changes and communicate those changes with
 Github

Why use version control (Git)?



If only I could go back to a version that was working...

Why use version control (Git)?

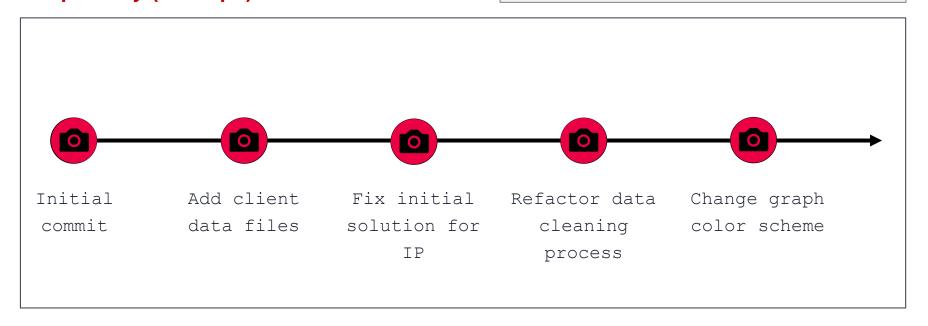


High-Level Idea

repository (aka repo)

Benefits:

- Git stores commit messages
- Snapshot of entire repo, not one file
- Commits are lightweight



Create a repository - Commands

1. Create a new directory for your repo

mkdir <directory name>

2. Navigate to the directory

cd <directory name>

3. Initialize repository (you only need to do this once for each repo you create)

git init

Initialize a Git Repo - Demo

Let's turn mydirectory into a git repo.

1. Navigate to a mydirectory if you aren't already there

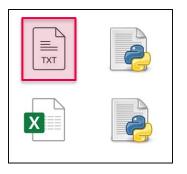
cd mydirectory

2. Initialize the git repo

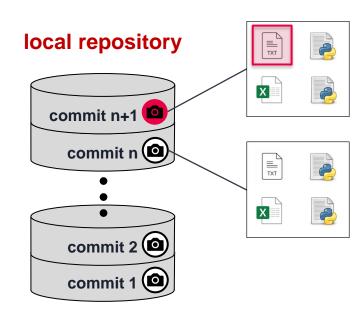
git init

Add and commit

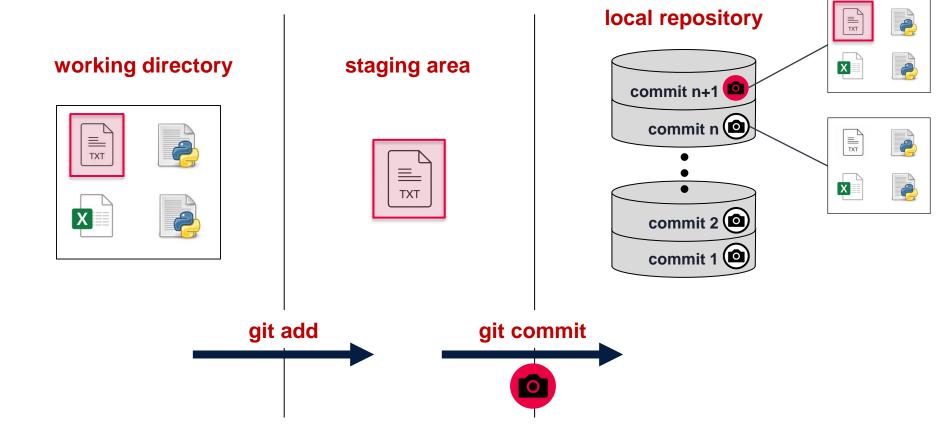
working directory







Add and commit



Add and commit - Commands

- 1. Make changes in working directory
- 2. Check the files in the staging area and any "untracked" changes (not yet added)

git status

3. As you change files: working directory → staging area

git add <filename>

git add . (adds all files)

4. Once you have added a new functionality/completed an update:
 staging area → local repo

git commit -m "comment"

Add a short, intuitive comment describing the change in functionality

→ For reference later!

Poll Question

pollev.com/alexandriaschmid171

In your opinion, which would be the most helpful commit message three months from now?

- A.) "Fix bug"
- B.) "Add functions: cleanData, processData, runPredictions"
- C.) "Add function to plot model results"
- D.) "Write latest optimization model"

Add and commit - Demo

Let's commit something to mydirectory! Run git status between each step to see the staging area.

1. Check the current status

git status

2. Add the file to the staging area

git add newfile.txt git status

3. Commit the changes to the repository

git commit -m "Add empty file" git status

Add and commit - Try it out

We want to create a new repository called myrepo (outside of mydirectory), create a text file in myrepo called myfile.txt, and add and commit it to the repository.

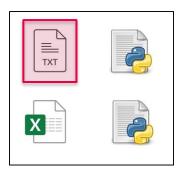
1. Leave mydirectory. Create and navigate to a new directory and initialize Git.

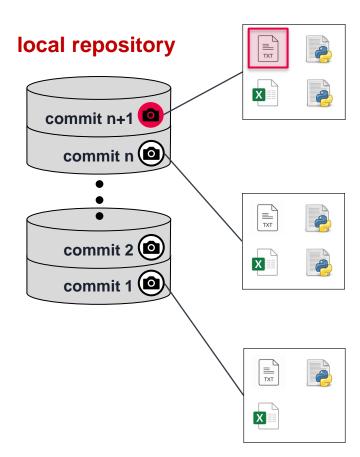
2. Create an empty text file in myrepo

3. Add and commit the text file to your new repository

Reverting Changes

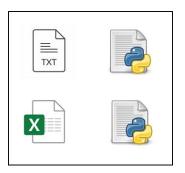
working directory

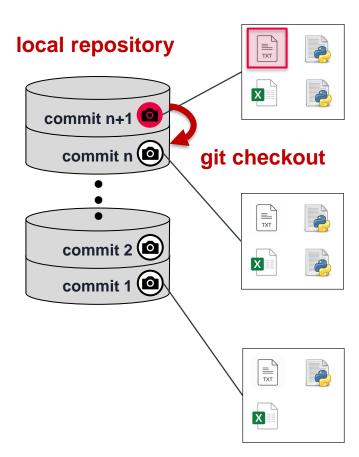




Reverting Changes

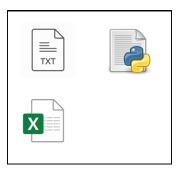
working directory

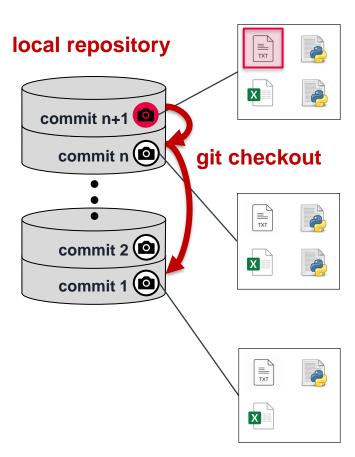




Reverting Changes

working directory





Reverting Changes - Commands

See log of commits, including SHA ID for each version

git log

Return to a previous version of a file from the commit history

git checkout <version_SHA>

git checkout <version_SHA> <filename>

See differences between two commits or differences in a specific file from two commits

git diff <version1_SHA> <version2_SHA>

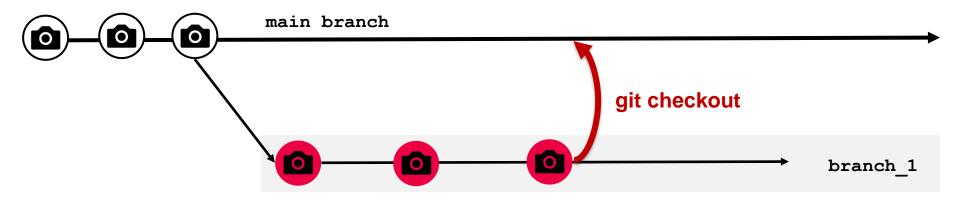
Reverting Changes - Try it out

Add some text to myfile.txt and commit. Check the log of commits, look at the differences between our two commits so far, and then check out the first commit.

1. Edit and add some text to myfile.txt and commit changes.

- 2. View the log to see our commit history
- 3. Find the differences between the two commits (The first 7 digits of the SHA id should be sufficient)
- 4. Check out the previous commit

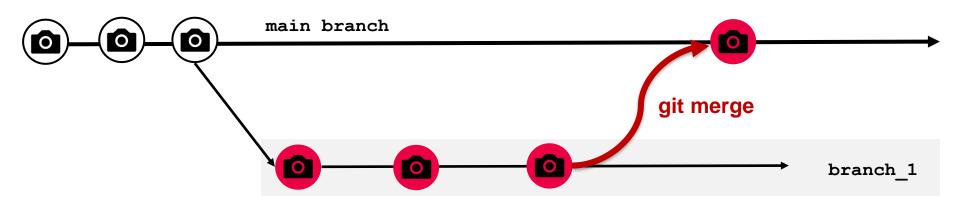
Branching



Suppose you run a process for a client every morning. You want to make a change, but it will take a few days. How can you work on the change AND run the stable version?

Branching

Branch when the intention is to eventually "merge" the branch back in with main



Suppose you run a process for a client every morning.
You want to make a change, but it will take a few days.
How can you work on the change AND run the stable version?

Branching and Merging - Commands

See all branches	git branch
Create and navigate to a new branch	git checkout -b new_branch_name
Switch between branches	git checkout branch_name
Merge two branches	git checkout branch_receiving_changes git merge branch_giving_changes
Delete a branch	git branch -d branch namo

Branching - Demo

Create a new branch called my_branch, add a file called mynewfile.txt to my branch and commit, and look at the list of branches and log of commits.

1. Create and check out a new branch (one step)

git checkout -b my branch

2. Add a new file to the new branch

touch mynewfile.txt

3. Add the new file to staging area and commit

git add mynewfile.txt
git commit -m "Add a second text file"

4. Display the branches, then look at the log of commits

git branch git log

Merging - Demo

Navigate to the main branch, merge my_branch into main, and delete my_branch.

1. Navigate to the main branch

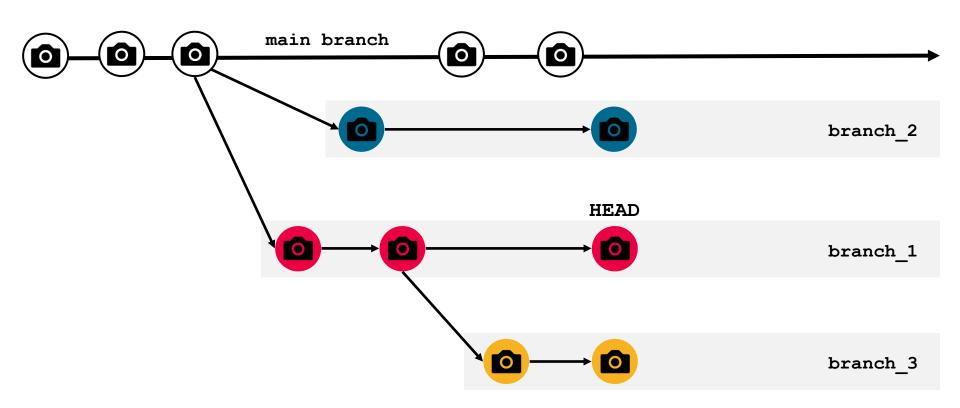
git checkout main

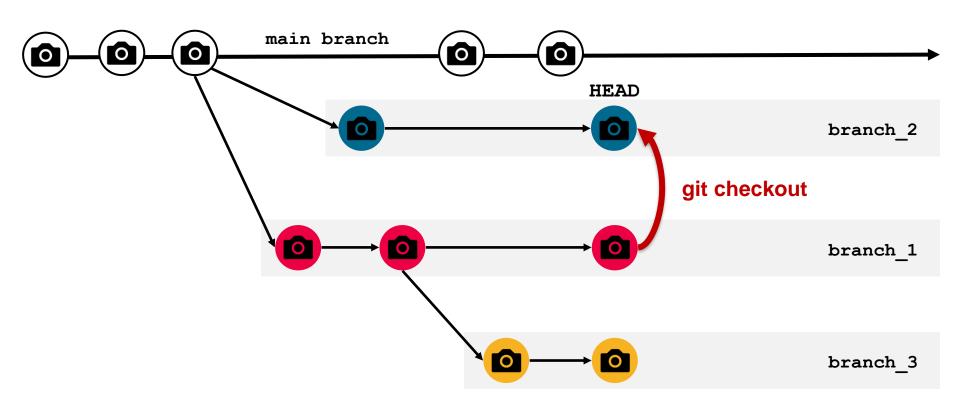
2. Merge my branch into the main branch

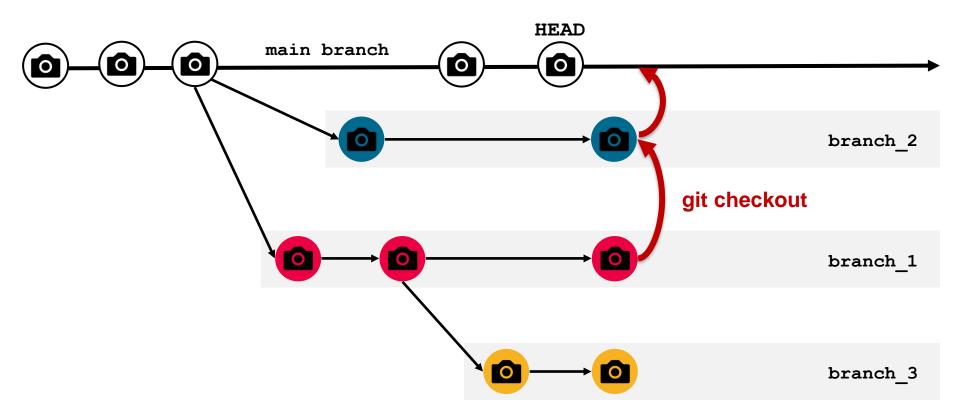
git merge my branch

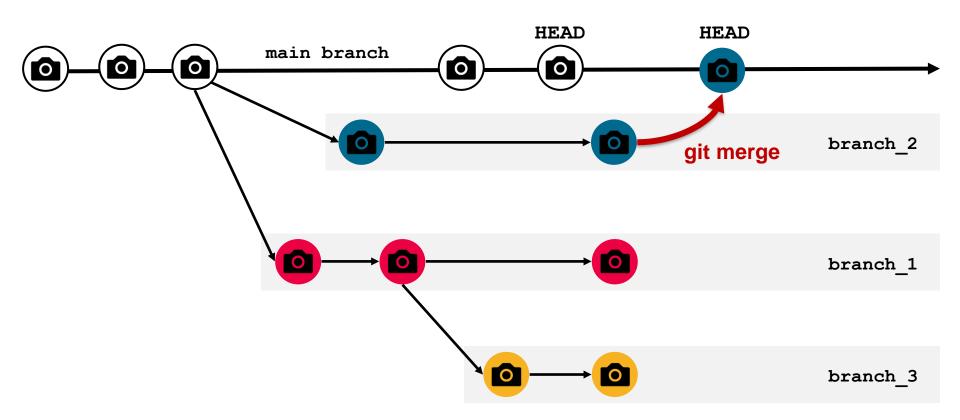
3. Delete my branch

git branch -D my branch



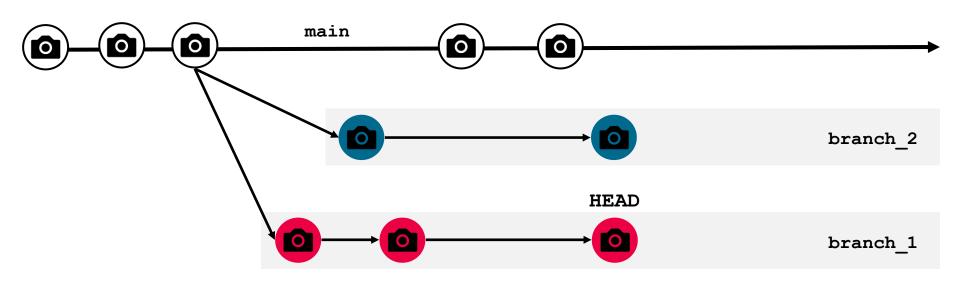






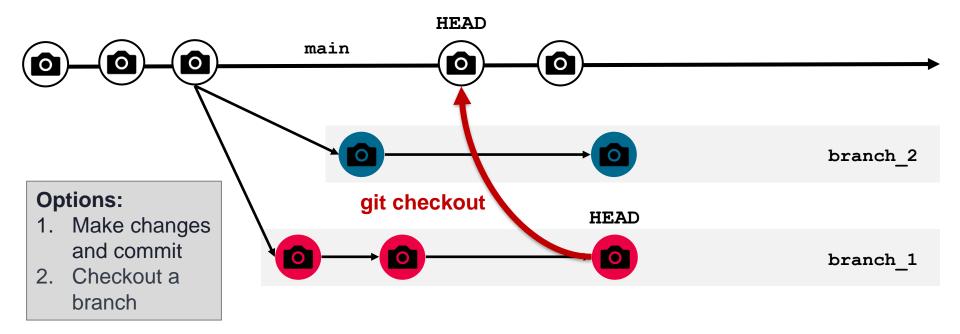
Aside: 'detached HEAD' state

HEAD usually points at a branch directly. When you checkout a commit, HEAD points instead points directly at that commit. This is a 'detached HEAD' state. That's okay!



Aside: 'detached HEAD' state

HEAD usually points at a branch directly. When you checkout a commit, HEAD points instead points directly at that commit. This is a 'detached HEAD' state. That's okay!



Branching - Try it out

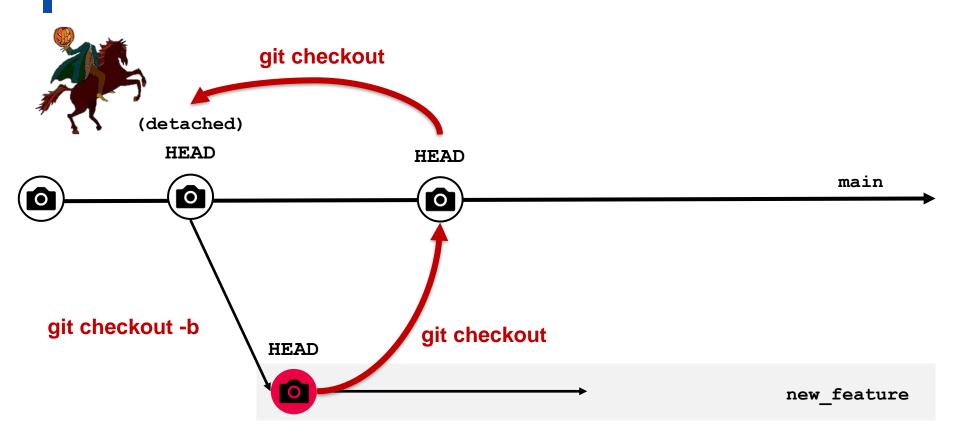
Checkout the main branch. Then checkout an earlier commit to enter a detached HEAD state. Create a new branch called new feature. Check out the main branch.

1. Checkout main, view log, and checkout an earlier commit

2. Create a new branch from this commit

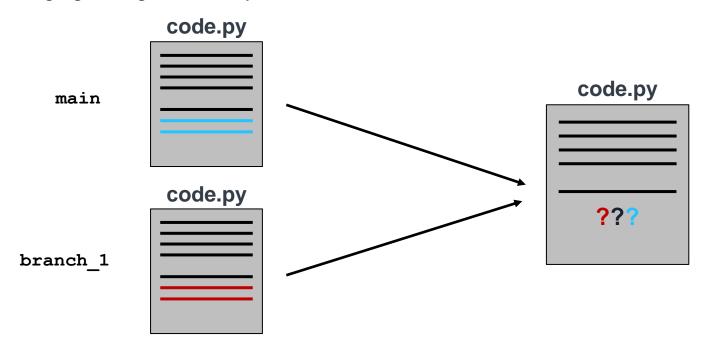
3. Display the branches, checkout main, then look at the log of commits

'detached HEAD' state



Merge Conflicts

When merging changes, we may sometimes find that our branches conflict.



Merge Conflicts

HEAD

The first line of the file is the same The second line is different

Commit dabb4c8c

The first line of the file is the same See? It's different



The first line of the file is the same
<<<<<< HEAD
The second line is different
======
See? It's different
>>>>>> dabb4c8c450e8475aee9b14b4383acc99f42af1d

When this happens, Git shows you exactly what the conflicts are and where they are located. It is up to you to reconcile the conflicts and commit them.

Merge Conflict - Demo (code along!)

Add text to myfile.txt in main and commit. Navigate to the new_feature branch, add different text to myfile.txt, and attempt to merge new feature into main.

1. Add text to myfile.txt and commit to main

2. Add different text to myfile.txt in new feature and commit

Merge Conflict - Demo (code along!)

Add text to myfile.txt in main and commit. Navigate to the new_feature branch, add different text to myfile.txt, and attempt to merge new feature into main.

3. Navigate to the main branch and merge new_feature branch

```
git checkout main
git merge new_feature
```

4. Resolve merge conflict

```
nano myfile.txt
```

5. Commit changes to main

```
git add myfile.txt
git commit -m "Resolve myfile conflict"
```

pollev.com/alexandriaschmid171

Suppose you create a new branch and add a file, branchfile.txt, and commit. You then add a file, mainfile.txt, to the main branch and commit. When you try to merge, your branch to main, what do you expect to happen?

- A.) Merge successful! Both files exist in main
- B.) Merge successful! But main now holds only branchfile.txt but not mainfile.txt
- C.) Merge conflict, you must resolve

pollev.com/alexandriaschmid171

You add a file, community.txt, to the main branch and commit. You then create a new branch and add a file, branchfile.txt, and commit. You then delete community.txt from main and commit. When you try to merge your branch to main, what do you expect to happen?

- A.) Merge successful! And branchfile.txt exists in main
- B.) Merge successful! But branchfile.txt does not exist in main
- C.) Merge conflict, you must resolve

Git Wrap-Up

- Commit often, write descriptive messages, and keep each commit to one "atomic" task
 - → Almost anything can be undone, as long as it's committed
- Take advantage of branching when you have new lines of development and don't want to disrupt the functionality of the main branch
- Google is your friend (e.g. "How to undo commit in Git".)

Break

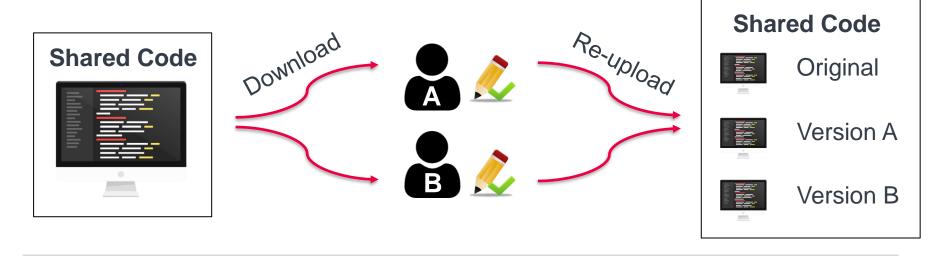
Resume in 10 minutes

Github

Adapted from slides by Galit Lukin and Jackie Baek with activities from Turing Way and Software Carpentry

Why use Github?

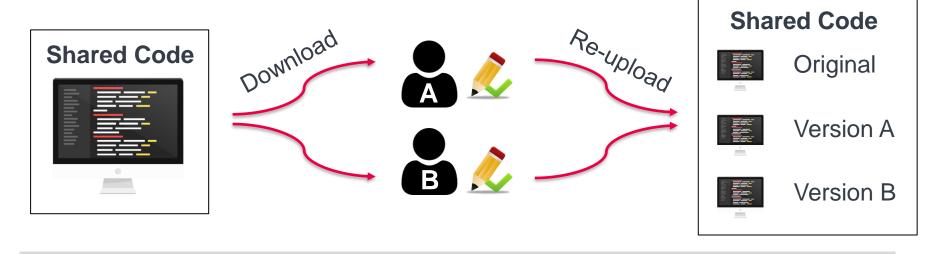
You're working on a research project with another graduate student and both have access to the code in a shared space, e.g. Dropbox.



What are some potential problems that could arise using this process?

Why use Github?

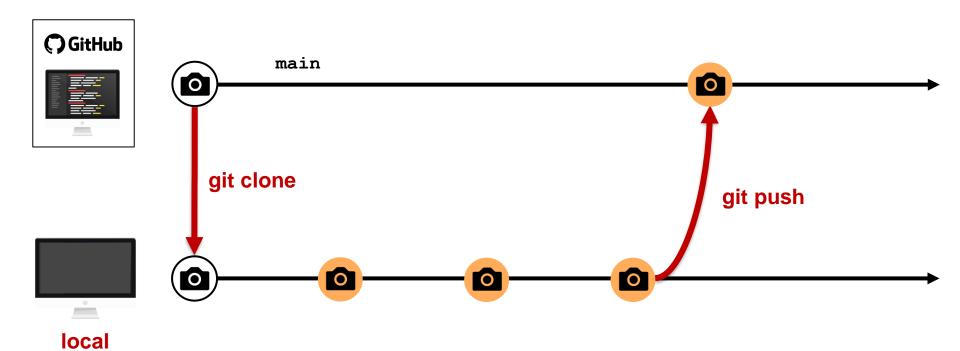
You're working on a research project with another graduate student and both have access to the code in a shared space, e.g. Dropbox.



Potential issues: Documenting all changes, staying up to date with changes, overwriting others' work, conflicts between versions, etc.

High-Level Idea

repository

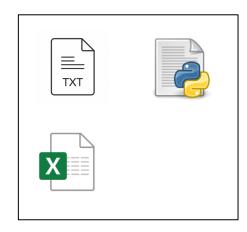






git remote add



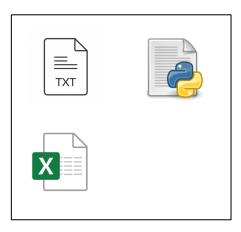


Your collaborator's local repository

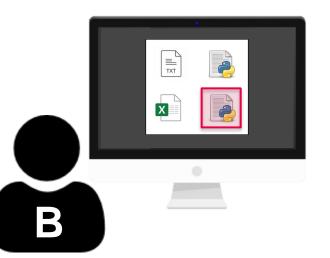






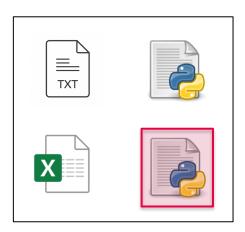


Your collaborator's local repository

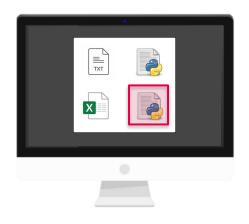






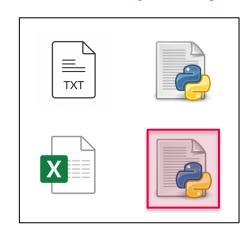


YOUR local repository









Local and Remote - Commands

Add a remote connection to a Github repo for an existing local repo

git remote add origin <Github SSH>

Clone existing Github repo

git clone <Github URL>

Pull from Github repo to local repo

git pull

Push from local repo to Github repo

git push origin
branch name>

Local machine's name for the remote repo (origin is usual convention)

Cloning a Github repo - Try it out

Clone the class repo into the location of your choice.

1. Navigate to wherever you cloned the repo (or already cloned the repo)

2. Clone the Github repository to your machine or pull changes if you already cloned it

3. Find it on your computer and check it out! You should see the module for this session (or change your directory and type 1s)

Creating a Github repo through Github

Create your own repository in Github called myrepo (we'll walk through this together).

- 1. Go to github.com and login
- 2. Click "New" in the top left by your list of repositories and fill in details

Top Repositories	☐ New
Find a repository	

3. Clone the Github repository to your machine [skip this step for now]

```
git clone https://github.com/<username>/<reponame>.git
```

Creating a Github repo for existing local repo

Create a remote connection for myrepo

1. Navigate to the existing repo (from the Git part of class)

cd myrepo

2. Add a remote connection

git remote add origin <Github SSH>

3. Push to upstream to Github

git push -u origin main

Pushing to Github - Demo

Create a new local branch called new_branch, add some text to mynewfile.txt on new branch and commit, merge the changes into main, and push to Github.

1. Create a new file and add a line of text

```
touch localfile.txt nano localfile.txt
```

2. Commit the changes locally

```
git add localfile.txt
git commit -m "Add file to local repo"
```

3. Push the changes to Github

```
git push origin main
```

Merge conflicts with Github

Your collaborators may have pushed new changes to Github while you've been working on your own changes locally.

Best practices:

- Pull from Github often, even when you aren't making changes to the remote repo
- Pull right before you push your changes
- Resolve merge conflicts as they arise when you pull

```
For example,

git commit -m "Fix bug"

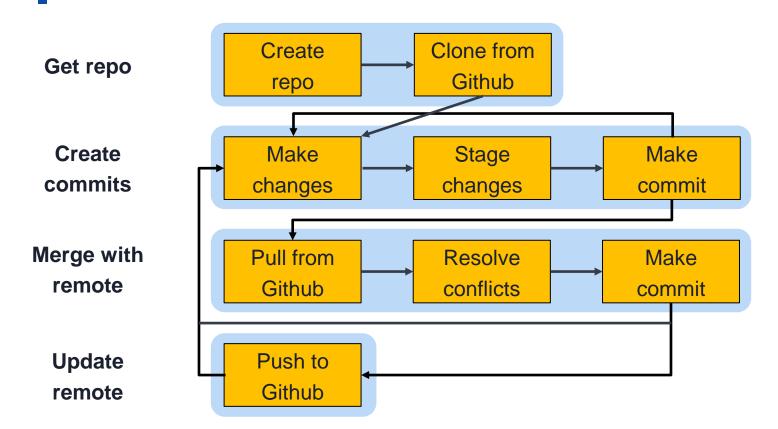
git pull

git push origin main
```

Putting it all together – Try it out

Create a new local branch called new_branch, add some text to newfile.txt on new_branch and commit, merge the changes into main, pull any new changes from Github, and push to Github.

Review of Basic Workflow



When do you make **Review of Basic Workflow** a commit? Think "atomic" Clone from Create Single bug fix, Get repo Github repo feature, re-factor Create Make Make Stage commits changes changes commit Merge with Pull from Resolve Make remote Github conflicts commit **Update** Push to remote Github

When do you push **Review of Basic Workflow** to Github? Completed and Clone from Create tested feature, Get repo Github repo ready for use or review Create Make Make Stage commits changes changes commit Merge with Pull from Resolve Make remote Github conflicts commit **Update** Push to remote Github

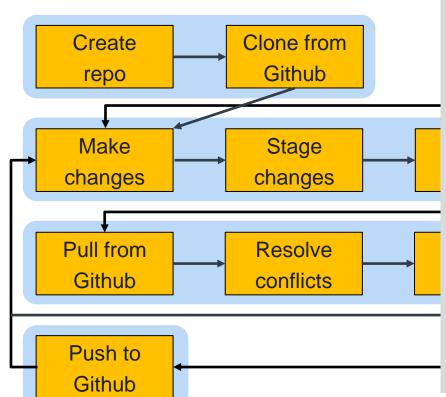
Review of Basic Workflow

Get repo

Create commits

Merge with remote

Update remote



When should I branch?

- May need to run main
- When you and a collaborator are working on different things

Why might I push my branch to Github?

- If you want to collaborate on the branch
- As a code back-up!

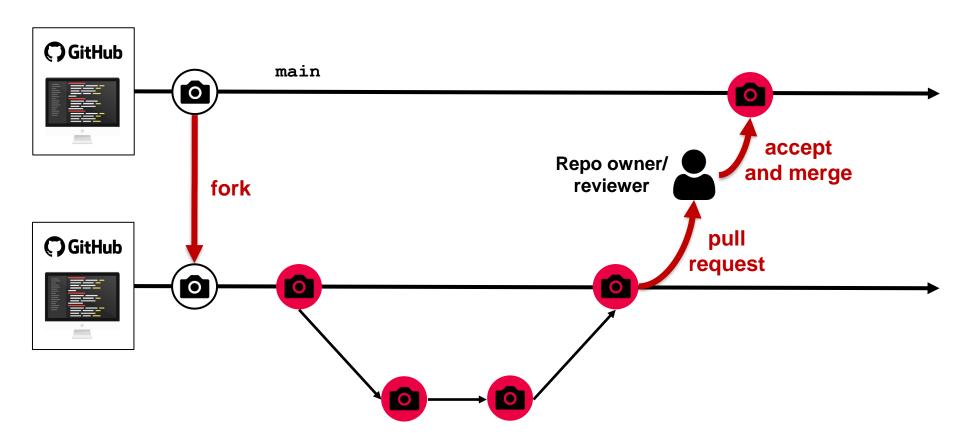
Social Coding on Github

Thus far, we've focused on using Github with a small group of people who have shared access to a repository.

How can you contribute to projects in the broader Github community?

- Not everyone has repo access, so we can't just push our changes
- Repository owner has a reviewing mechanism

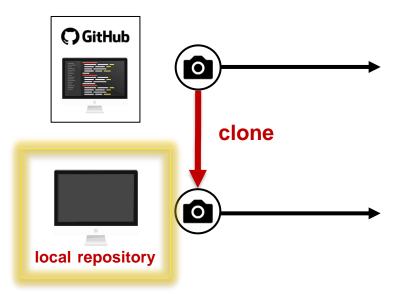
Another Workflow – Forking / Pull Requests



Cloning vs. Forking

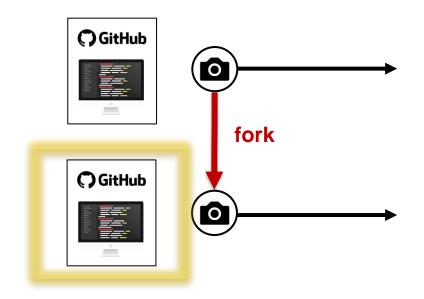
Clone

- Creates a local copy of the repo
- Execute in terminal with git



Fork

- Creates a copy of the repo on Github
- Execute in Github



How to fork a repo on Github

Fork a repository from Github and clone your forked copy onto your local machine.

- 1. Go to github.com, login, and find the repo you want to clone
- 2. Click "Fork" in the right of the repo page

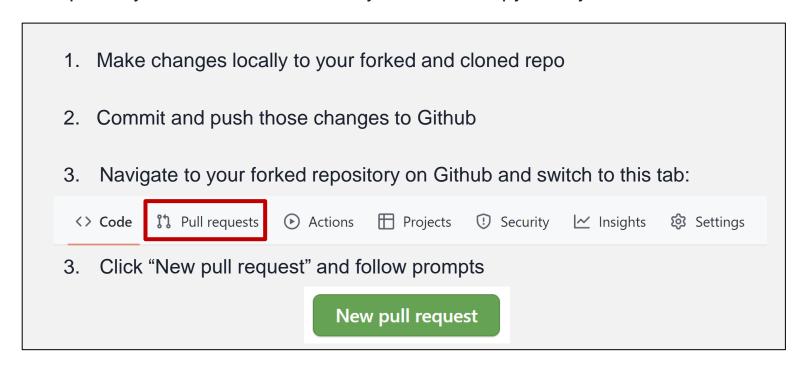


3. Clone the forked Github repository to your machine

git clone https://github.com/<yourusername>/<reponame>.git

How to submit a pull request on Github

Fork a repository from Github and clone your forked copy onto your local machine.



pollev.com/alexandriaschmid171

Your collaborator gave you access to a repository that contains shared code. You want to be able to make your own changes and push them to the Github repo.

Should you clone or fork the repository?

- A.) Clone
- B.) Fork

pollev.com/alexandriaschmid171

You found a Github repo and think it would be a great starting point for a project you're working on.

Should you clone or fork the repository?

- A.) Clone
- B.) Fork (then clone)

pollev.com/alexandriaschmid171

You find a repo on Github that has some room for improvement, but to which you don't have access.

What would happen if you tried to clone the repository?

- A.) An error. You can't clone without access.
- B.) Git would create a local copy of the repo, but you couldn't push changes.
- C.) Git would create a local copy of the repo and you could push changes to the Github repo.

Putting it all together – Task 1

Fork alexschmid3/putitalltogether and clone it to your local machine. Add a text file with a unique name that contains your name (and a fun fact?) to the students folder and commit. Push changes to your forked repo, then send a pull request to me.

- 1. Fork repo on Github
- 2. On your local machine:

Putting it all together – Task 1

Fork alexschmid3/putitalltogether and clone it to your local machine. Add a text file with a unique name that contains your name (and a fun fact?) to the students folder and commit. Push changes to your forked repo, then send a pull request to me.

3. Submit a pull request to me

Putting it all together – Task 2

Get a list of the contents of the students folder and send the output to outputs/studentlist.txt. What do you notice when you run git status? What happens when you try to add and commit changes?

Hint:

Print out the contents of .gitignore file. What do you think this file's purpose is?