QSS20: Modern Statistical Computing

Unit 05: Workflow tools

Agenda

- Review of final project
- Recap of user-defined functions
- Basic command line syntax
- Git/GitHub

Agenda

- **Review of final project**
- Recap of user-defined functions
- Basic command line syntax
- Git/GitHub

Final project components

- **Milestone 1 due 02/12:** memo or plan for what question you'll ask and analyses you'll run
- **Milestone 2 due 02/26:** set up your repository and start coding
- **Final outputs due 03/14** (see course website for more details):
  - Final presentation done in Beamer (LaTeX-based slides software) via Overleaf
  - Short 10-page report (also done in LaTeX)
  - Github repo and readme with all code to reproduce analyses

Final project options

1. Social Impact Practicum (SIP) 1: Medical training data
2. SIP 2: Large dataset on START participants (SIRS)
3. Cook Count sentencing data
4. Independent project

SIP option 1: Medical training data

Data:

- 6-hour training for medical students at multiple schools
- 15 modules with 6 questions each, both multiple choice and open-ended/qualitative, evaluating:
  - Overall satisfaction with training
  - What they found was helpful
  - Shifts in their knowledge of/attitudes toward IDD
- Composition challenges: Unable to track subjects over time

Questions:

- **Is this working?**
  - Changes in perspectives and depth of understanding toward IDD?
  - Consider training outcomes from ranking questions (e.g., with regression) and free-form text (e.g., topic models)
  - Connect with participant demographics
- **What training components matter most?**
  - Expert presentation & best practices
  - Guest speakers with personal experience of IDD
  - Other training elements suggested in open-ended questions

SIP option 2: SIRS

Data: High-risk START participants: millions of records, 1̃3,000 people from 2013 to 2021

- These include:
    - Encounters with law enforcement
    - Emergency visits
    - Physical restraint during crises
    - Demographics
    - Intake info

Questions:

- Inequalities among START participants by race, gender, and region?
- Could consider frequency, duration, and outcomes of such events
- Could relate them to social isolation (length of time since beginning of COVID-19 pandemic as a proxy)

Q&A

What questions do you have?

“Project shopping”

Goal: Connect with classmates around project ideas

Group by option you’re most interested in, and **feel free to float around!**
From student orientation:

- Left side of room: SIP large dataset (SIRS)
- Right side of room: SIP medical training
- Back of room: Cook County sentencing dataset
- Front of room: Independent project

Where we are

- Review of final project
- **Recap of user-defined functions**
- Basic command line syntax
- Git/GitHub

Recap of user-defined functions

What do you remember from last class?

Recap of user-defined functions

Tips:

- Lambda functions: single-use, quick data transformations
- User-defined functions: re-usable, easier to document & read
  - Groups of functions: can post to PyPI to benefit others
- Ingredients: Name & inputs; meat/workhorse; return statement
- Workflow: Code for example → generalize key features (e.g., Ward #) → build function → test on examples

Useful code snippets:

```
df[col].apply(lambda row: row.split()[0]) # get first token

def summarize_nearby_crimes(crime_num: str, days_num: int):
    '''Description... Params... Returns... '''
    df = df[df.CCN == crime_num] # filter by crime
    samew = compare[compare.WARD==df.WARD].copy() # filter by ward
    pct = buff[buff.OFFENSE==df.OFFENSE]/buff.shape[0] # % same
    return(pct)
```

Where we are

- Review of final project
- Recap of user-defined functions
- **Basic command line syntax**
- Git/GitHub

Why are we covering this?

- **Easiest way to interface with Git/GitHub:** as we'll discuss next, Git/GitHub have a graphical user interface (GUI), or a way to go to a website and point/click, but that defeats a lot of the purpose
- **Moving files around on jupyter hub**
- **Interacting with high-performance clusters/long-running jobs:** a lot of what we'll be doing is code written in jupyter notebooks (.ipynb) that runs relatively quickly; for your projects you may want to run `.py` scripts or use high-performance computing

Where is the "command line" or what's a terminal?

- On Mac/OSX or Linux, terminal is native! You can find it by opening up spotlight and searching for terminal
- On Windows, this takes more work. Options include:
  - Installing Ubuntu (see Windows store)
  - Installing git bash (lightweight)
  - See more info on the course page

First set of commands: navigating around directory structure

1. Where am I?
   `pwd`
2. How do I navigate to folder *foldername*?
   `cd foldername`
3. I'm lost; how do I get back to the home directory?
   `cd`
4. How do I make a new directory with name *foldername*?
   `mkdir foldername`
5. What files & directories are in this dir? (see more sorting options)
   `ls`
   `ls -t`
6. How do I navigate "up one level" in the dir structure?
   `cd ../`

Activity (on your terminal/terminal emulator)

1. Open up your terminal
2. Navigate to your `Desktop` folder
3. Make a new folder called `qss20_clfolder`
4. Within that folder, make another subfolder called `sub`
5. Enter that subfolder and list its contents (should be empty)
6. Navigate back up to `qss20_clfolder` without typing its full pathname

Second set of commands: moving stuff around

1. Create an empty file (rarer but just for this exercise)
   ```
   touch examplefile.txt
   ```
2. Copy a specific file in same directory (more manual)
   ```
   cp examplefile.txt examplefile2.txt
   ```
3. Copy a specific file in same directory and add prefix (more auto):
   ```
   for file in examplefile.txt; do cp ‘‘$file’’ ‘‘copy_$file’’;
   done
   ```
4. Move a file to a specific location (removes the copy from its orig location; root path differs for you)
   ```
   mv copy_examplefile.txt /Users/jhaber/Desktop/qss20_clfolder/
   ```
5. Move a file “down” a level in a directory
   ```
   mv copy_examplefile.txt sub/
   ```
6. Move a file “up” one level
   ```
   mv copy_examplefile.txt ../
   ```
7. Up two levels:
   ```
   ../../
   ```

Third set of commands: deleting things

1. Delete a file

   `rm examplefile.txt`

2. Delete a directory

   `rm -R examplefolder`

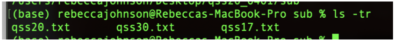
3. Delete all files with a given extension (example deleting all pngs; can use with any extension)

   `rm *.png`

4. Delete all files with a specific pattern (example deleting all files that begin with phrase testing)

   `rm testing*`

5. Can do more advanced regex- eg, deleting all files besides the qss20 one in this dir

   ```
   /Users/rebeccajohnson/Desktop/qss20_0401/sub
   (base) rebeccajohnson@Rebeccas-MacBook-Pro sub % ls -tr
   qss20.txt       qss30.txt       qss17.txt
   (base) rebeccajohnson@Rebeccas-MacBook-Pro sub %
   ```

   ```
   find sub/ -name 'qss[1|3][7|0].txt' -delete
   ```

Live coding @ command line

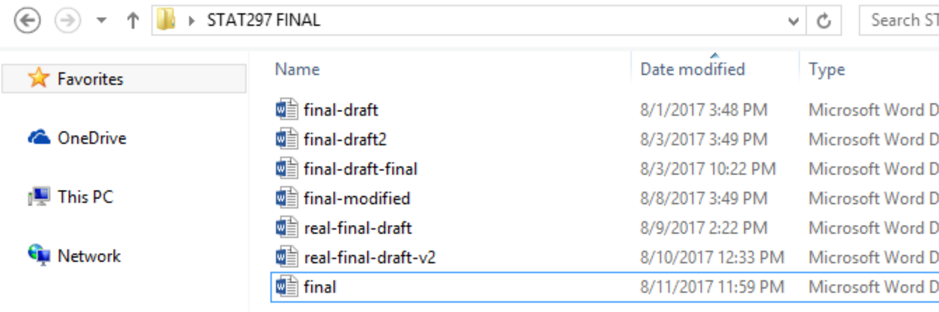
Command line activity

1. Delete the `sub` directory in `qss20_clfolder`
2. Use touch to create the following two files in the main `qss20_clfolder`:
   `00_load.py`
   `01_clean.py`
3. Create a subdirectory in that main directory called `code`
4. Move those files to the `code` subdirectory without writing out their full names
5. Copy the `01_clean.py` into the same directory and name it `01_clean_step1.py`
6. Remove all files in that directory with clean in the name

Where we are

- Review of final project
- Recap of user-defined functions
- Basic command line syntax
- **Git/GitHub**

Motivation for Git/GitHub



Name	Date modified	Type
final-draft	8/1/2017 3:48 PM	Microsoft Word D
final-draft2	8/3/2017 3:49 PM	Microsoft Word D
final-draft-final	8/3/2017 10:22 PM	Microsoft Word D
final-modified	8/8/2017 3:49 PM	Microsoft Word D
real-final-draft	8/9/2017 2:22 PM	Microsoft Word D
real-final-draft-v2	8/10/2017 12:33 PM	Microsoft Word D
final	8/11/2017 11:59 PM	Microsoft Word D

Source: SMAC group

What is Git?

- Set of command line tools for version control (aka avoid finalfinal, finalrealthistime, etc.)
- “Distributed”: rather than stored centrally in one place, files/code can be stored on all collaborators’ machines
- git for command line/regular use, GitHub for online interface/sharing code publicly

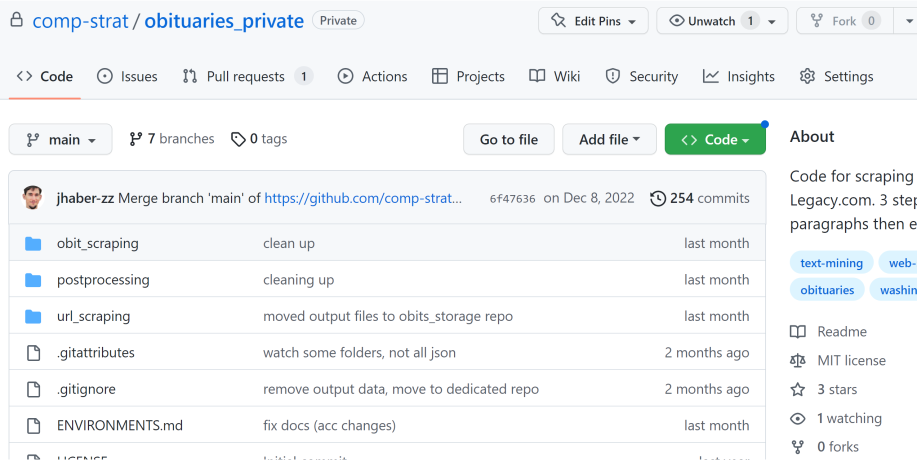
What is GitHub?

- Web-based repository for code that utilizes `git` version control system (VCS) for tracking changes
- Has additional features useful for collaboration, some of which we'll review today (repos; issues; push/pulling recent changes) and others of which we'll review as the course progresses (branches; pull requests)
- Why GitHub rather than Dropbox/google drive?
  - Explicit features that help with simultaneous editing of the same file
  - Public-facing record, or a portfolio of code/work (if you make it public)
  - Ways to comment on and have discussions about code specifically through the interface

Example repo: private repo

https://github.com/comp-strat/obituaries_private

If you go to this url, get 404 error unless you're added as a collaborator



comp-strat / obituaries_private Private

jhaber-zz Merge branch 'main' of https://github.com/comp-strat... 6f47636 on Dec 8, 2022 254 commits

obit_scraping	clean up	last month
postprocessing	cleaning up	last month
url_scraping	moved output files to obits_storage repo	last month
.gitattributes	watch some folders, not all json	2 months ago
.gitignore	remove output data, move to dedicated repo	2 months ago
ENVIRONMENTS.md	fix docs (acc changes)	last month
LICENSE	Initial commit	last year

About

Code for scraping Legacy.com. 3 step paragraphs then e

text-mining web- obituaries washin

Readme

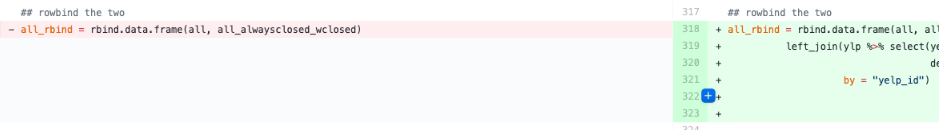
MIT license

3 stars

1 watching

0 forks

Example: tracked changes in code when you "push" updated version

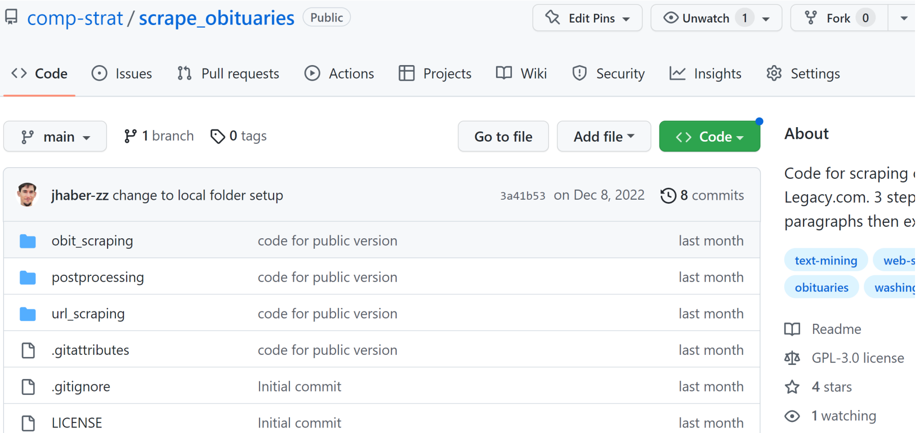


```
  ## rowbind the two
- all_rbind = rbind.data.frame(all, all_alwaysclosed_wclosed)
```

```
  ## rowbind the two
+ all_rbind = rbind.data.frame(all, all
+          left_join(ylp %>% select(ye
+                                    de
+                    by = "yelp_id")
+
+
```

Example repo: public repo

Codebase for scraping obituaries for Washington, DC (feel free to poke around, make issues, etc.): https://github.com/comp-strat/scrape_obituaries



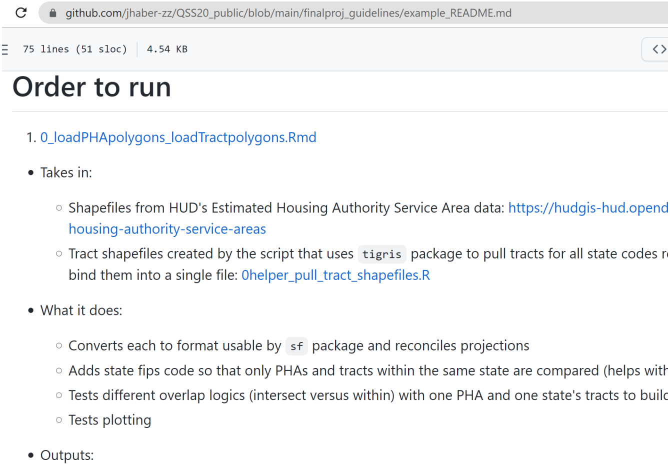
jhaber-zz change to local folder setup 3a41b53 on Dec 8, 2022 8 commits

obit_scraping	code for public version	last month
postprocessing	code for public version	last month
url_scraping	code for public version	last month
.gitattributes	code for public version	last month
.gitignore	Initial commit	last month
LICENSE	Initial commit	last month

Code for scraping
Legacy.com. 3 step
paragraphs then e

Ingredients of a repo: README

- Should provide project description; purpose, inputs, outputs of each script
- Might also have installation instructions, directions on where to download data, etc.
- See example in course repo under `finalproject_guidelines/`



github.com/jhaber-zz/QSS20_public/blob/main/finalproj_guidelines/example_README.md

75 lines (51 sloc) | 4.54 KB

Order to run

1. 0_loadPHApolygons_loadTractpolygons.Rmd

- Takes in:
  - Shapefiles from HUD's Estimated Housing Authority Service Area data: https://hudgis-hud.opend
    housing-authority-service-areas
  - Tract shapefiles created by the script that uses `tigris` package to pull tracts for all state codes r
    bind them into a single file: 0helper_pull_tract_shapefiles.R
- What it does:
  - Converts each to format usable by `sf` package and reconciles projections
  - Adds state fips code so that only PHAs and tracts within the same state are compared (helps with
  - Tests different overlap logics (intersect versus within) with one PHA and one state's tracts to builc
  - Tests plotting
- Outputs:

Ingredients of a repo: directories

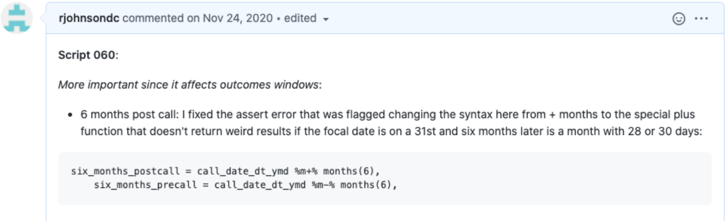
Command line syntax in previous slide is useful for org/reorg. For our class, we'll generally have two directories:

1. `code/` (with subdir for tasks)
2. `output/` (with subdir for tables versus figs)

Depending on the context, you *may* store data, but (1) GitHub has file size limits (100 MB max), and (2) sensitive data should generally not be put in a repo, even if the repo is private (instead, read directly directly from its source or have download instructions)

Ingredients of a repo: issues

- Can assign to specific collaborators or leave as a ”note to self”to look back at something
- Can use checklist features
- Can include code excerpts
- Easy to link to a specific commit (change to code)
- Need to be logged into GitHub to write



rjohnsondc commented on Nov 24, 2020 • edited

Script 060:

More important since it affects outcomes windows:

- 6 months post call: I fixed the assert error that was flagged changing the syntax here from + months to the special plus function that doesn't return weird results if the focal date is on a 31st and six months later is a month with 28 or 30 days:

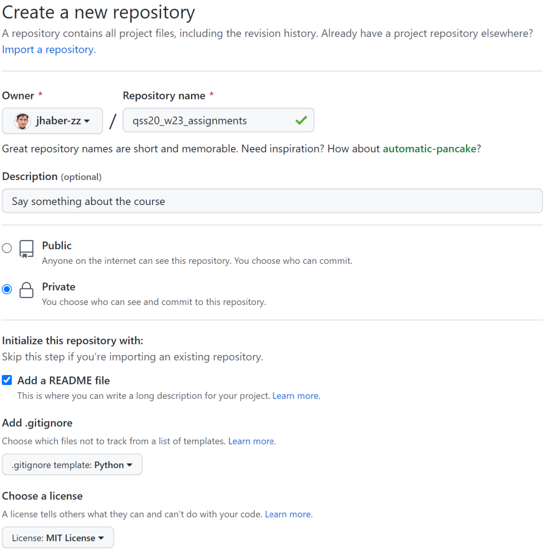
```
six_months_postcall = call_date_dt_ymd %m+% months(6),
    six_months_precall = call_date_dt_ymd %m-% months(6),
```

General steps in workflow

1. Create or clone a repository to track
2. Make changes to code or other files
3. **Commit** changes: tells the computer to “save” the changes
4. **Push** changes: tells the computer to push those saved changes to github (if file exists already, will overwrite file, but all previous versions of that file are accessible/retrievable)

How to create a new repository

- On GitHub.com, click "Repositories" then the green "new" button
- Enter a name (for command line reasons, avoid spaces)
- Give a brief description
- Initialize with a README
- Add a Python-specific `.gitignore` to help git focus
- Select a License (permissive is good)



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * jhaber-zz / Repository name * qss20_w23_assignments ✓

Great repository names are short and memorable. Need inspiration? How about automatic-pancake?

Description (optional): Say something about the course

○ Public
Anyone on the internet can see this repository. You choose who can commit.

● Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☑ Add a README file
This is where you can write a long description for your project. Learn more.

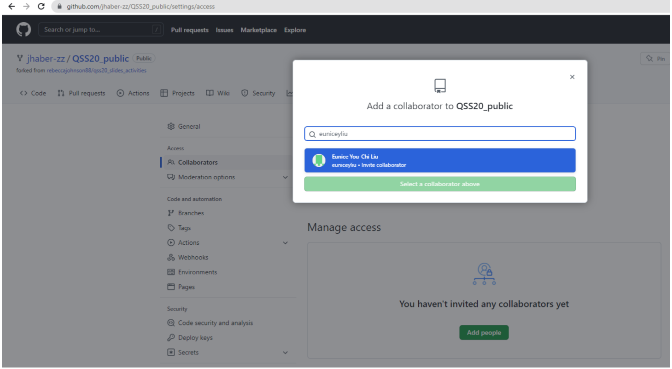
Add .gitignore
Choose which files not to track from a list of templates. Learn more.

.gitignore template: Python

Choose a license
A license tells others what they can and can't do with your code. Learn more.

License: MIT License

How to give write access to a repo



Add a collaborator to **QSS20_public**

euniceyliu

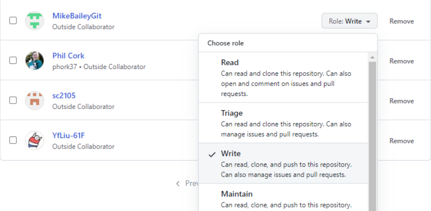
Eunice You-Chi Liu
euniceyliu • Invite collaborator

Select a collaborator above

Manage access

You haven't invited any collaborators yet

Add people



□ MikeBaileyGit
Outside Collaborator

Role: Write ▾ Remove

□ Phil Cork
phork37 • Outside Collaborator

Remove

□ sc2105
Outside Collaborator

Remove

□ YfLiu-61F
Outside Collaborator

Remove

Choose role

Read
Can read and clone this repository. Can also open and comment on issues and pull requests.

Triage
Can read and clone this repository. Can also manage issues and pull requests.

✓ Write
Can read, clone, and push to this repository. Can also manage issues and pull requests.

Maintain
Can read, clone, and push to this repository.

Contribute to a repository

1. Clone repo
2. Edit files (e.g., via Jupyter)
3. Send changes to Github

```
git status
git add notebook.ipynb data.csv # careful with `-A'!
git commit -m "this is what i changed"
git push
```

4. Send changes to GitHub (batch commits thoughtfully, often by file type; e.g., you created a bunch of figures that you want to push)

```
git status
git add *png
git commit -m "new figs"
git push
```

Live coding @ git

Focusing on first step: how to clone

1. Open your local terminal and navigate to where you want the repo's files to be stored
2. Go to GitHub.com and go to `Code` button to find the name of the repo
3. Type the following command to clone (reponame.git will be the name of the url you copy/pasted)

   ```
   git clone reponame.git
   ```

Activity 1: clone the public class repo so you can get recent changes

1. Open up terminal
2. Type:
   `git clone https://github.com/jhaber-zz/QSS20_public.git`
3. Use `cd` to navigate to `activities`
4. Open up a notebook and try editing an activity
5. Try using the `mv` command to move a blank problem set (e.g., `pset2_blank.ipynb`) to a different directory

Activity 2: create a private repo to submit your next problem set

1. Create a new private repo on GitHub using the website and instructions above; name it `qss20_w23_assignments`
2. Add Prof. (`jhaber-zz`) & (`euniceyliu` and `ramseyash`) as collaborators via GitHub
3. Clone the repo locally using your terminal/terminal emulator
4. Create two subdirectories: `code/` and `output/`
5. Within the `code/` subdirectory, move a file you have from another directory to that directory (e.g., .py, .R, .ipynb) or use `touch` to create blank file
6. Within the `output/` subdirectory, use `touch` to create a blank file
7. Push the changes you made to both subdirectories (requires personal access token)
8. Assign Prof. Haber & TAs an issue
9. **Congrats!** You just used git to develop code and submit something!

Activity 3: Create a git conflict

1. Using GitHub, edit the README to link to the changes you just made
2. *Without doing a git pull*, use your terminal to locally change (with nano or another text editor) some file other than README (e.g., could edit the text file or add a comment to the code file)
3. Try pushing your local changes. You should receive an error asking you to git pull first (may need to set merge method first)
4. Do a git pull and consolidate your changes with the remote, then git push
5. Try again: edit README on GitHub, edit it locally (without pulling), then try to push. To fix this, you can google (I often do) or for a hint, start with `git reset` or `git stash`

For that last step...

I SEE THAT YOU COMMIT YOUR CODE WITHOUT DOING A PULL FROM GIT TO GET THE LATEST UPDATES
I ALSO LIKE TO LIVE DANGEROUSLY
quickmeme.com

Problem set three submission instructions

- Write your problem set in one of these ways:
  - Locally: move the blank problem set to the `code` directory of the repo you created; edit there
  - Jhub: copy the blank problem set from `shared/QSS20_public/problemsets` folder to your own folder; edit there
  - Use Google Colab or some other cloud service with which you're already familiar
- In any case, store the file in the `code` directory of the repo you just created
- While working on the problem set, regularly repeat the git add, git commit, git push steps to get used to process and create tangible commits (e.g., "Completed first section")
- When you're ready for it to be graded, push two files to your repo (the raw .ipynb and compiled .html) AND assign me & TAs a GitHub issue to grade

How to collaborate on code with classmates

- Jupyter notebooks not an ideal collaborative tool, not built for version tracking
  - Don't allow simultaneous live editing, like google docs
  - Even with a shared virtual machine (e.g., Colab), interface is clunky: someone edits → everyone else immediately gets popup asking to overwrite their own version
- Suggestion 1: Live code collaboration sessions (at least two per pset)
  - Work through problems together in-person (ideally) or over zoom
  - Coordinate your schedules and plan ahead
- Suggestion 2: Work from **one group partner's** private git repository
  - Give the other person access and write permissions (see next slide)
  - Pass the editing baton back and forth, e.g.: "Hey, I'm done with section 1.3, want to pull my changes and start on 1.4?"
  - Once submitted, copy final version to other group repos (for reference)

Additional GitHub topics for another time

- Storing your credentials
- Tools for more collaborative coding: branching and pull requests
- Options to reverse changes
- Large file storage

Wrapping up

We covered:

- Basic command line syntax
  - Navigating around directory structure
  - Moving stuff around
  - Deleting things
- Git/GitHub
  - Git/GitHub workflow
  - Cloning the public class repo
  - Creating a private repo for pset submission