QSS20: Modern Statistical Computing

Unit 04: Workflow tools

Updated peer tutoring locations

- Wednesday 9/21 9-10pm: BERRY 367 (on 3fb)
- Thursday 9/22 7-8pm: BAKER 158 (baker mezzanine, off blobby, 1902 end)
- Silsby room requests in process

Goal for today's session

- Final projects:
  - Presentation by SIP lead Ashley Doolittle, Q&A
  - "Project shopping"
- Piazza hello
- Basic command line syntax
- Git/GitHub
- User-defined functions: lecture & activity from previous lecture

Goal for today's session

- **Final projects:**
  - **Presentation by SIP lead Ashley Doolittle; Q&A**
  - "Project shopping
- Piazza hello
- Basic command line syntax
- Git/GitHub
- User-defined functions: lecture & activity from previous lecture

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

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, $\tilde{1}$3,000 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)

Goal for today's session

- **Final projects:**
  - Presentation by SIP lead Ashley Doolittle; Q&A
  - **"Project shopping"**
- Piazza hello
- Basic command line syntax
- Git/GitHub
- User-defined functions: lecture & activity from previous lecture

Goal for today's session

- Final projects:
  - Presentation by SIP lead Ashley Doolittle; Q&A
  - "Project shopping
- **Piazza hello**
- Basic command line syntax
- Git/GitHub
- User-defined functions: lecture & activity from previous lecture

Piazza hello

What is this again?

- The main channel for course communication
- For Q&A, announcements, chat with project groups, etc.

What to expect:

- Prof and/or TA will respond within 24 hours
  - Reminder: Direct messages/emails to Prof. only for family emergencies and other personal issues
- **Before a problem set is due**, we will respond to all questions posted before **3 pm** on due date but not questions between 3 pm and midnight when due
- Please do suggest answers to others' questions!

Piazza hello

How to access:

- Click here to join class Piazza
- Then you can access it from course page on Canvas

Activity:

- Access now and post a note in the "greetings" folder
- Once you've done that, reply to a classmate's greeting from same folder

Where we are

- Final projects:
  - Presentation by SIP lead Ashley Doolittle, Q&A
  - "Project shopping
- Piazza hello
- **Basic command line syntax**
- Git/GitHub
- User-defined functions: lecture & activity from previous lecture

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**
- **TBD: 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; if we have time to cover high-performance computing, running `.py`

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

Let's practice!

Wherever your terminal is, open one up now!

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 and directories are in this directory? (many more sorting options here: https://man7.org/linux/man-pages/man1/ls.1.html)
   ```
   ls
   ls -t
   ```

6. How do I navigate "up one level" in the dir structure?
   ```
   cd ../
   ```

Activity (on your terminal/terminal emulator)

1. Find 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

1. Delete a file

   `rm examplefile.txt`

2. Delete a directory

   `rm -R examplefolder`

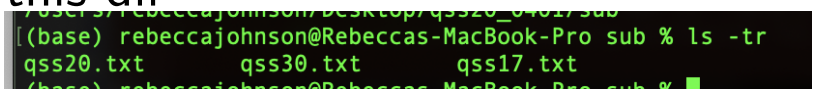
3. Delete all files with a given extension (example deleting all pngs; can use with any extension)

   `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



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

Activity (on your terminal/terminal emulator)

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

Introducing jupyter hub (jhub) and applying these commands

Jhub: cloud server similar to CoLab; each time it restarts it pulls latest materials from our `qss20_slides_activities` repo; way to access materials once we move away from Canvas posting

1. Navigate to `https://jhub.dartmouth.edu` and click on QSS20 option
2. Shared course materials (slides; in-class activities) are in the following read-only folder (shared/QSS20-22W/):

3. In the main directory (one level up from shared or the folder icon), create a folder to store editable files: `qss20_mywork`

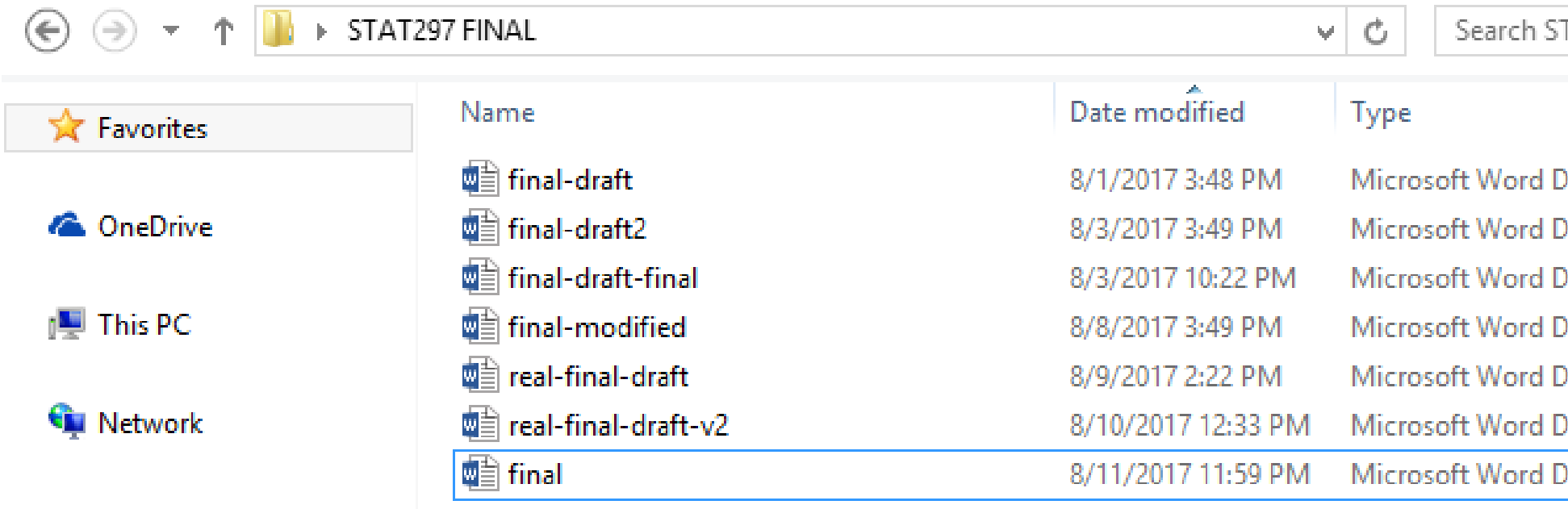
Activity (on jhub)

1. Navigate to the terminal via New $\Longrightarrow$ Terminal
2. If you haven't already, use `mkdir` to create a new directory `qss20_mywork`
3. Copy the following file from "shared/QSS20_public/activities/" into that directory:
   $01_{f}unctions_{b}lank.ipynb(ifit'snotshowingupgotocontrolpanelandrestartker$
4. Rename that file with your netid as a suffix before the .ipynb
5. Practice editing

Where we are

- Final projects:
  - Presentation by SIP lead Ashley Doolittle, Q&A
  - "Project shopping
- Piazza hello
- Basic command line syntax
- **Git/GitHub**
- User-defined functions: lecture & activity from previous lecture

Motivation for Git/GitHub



Source: SMAC group

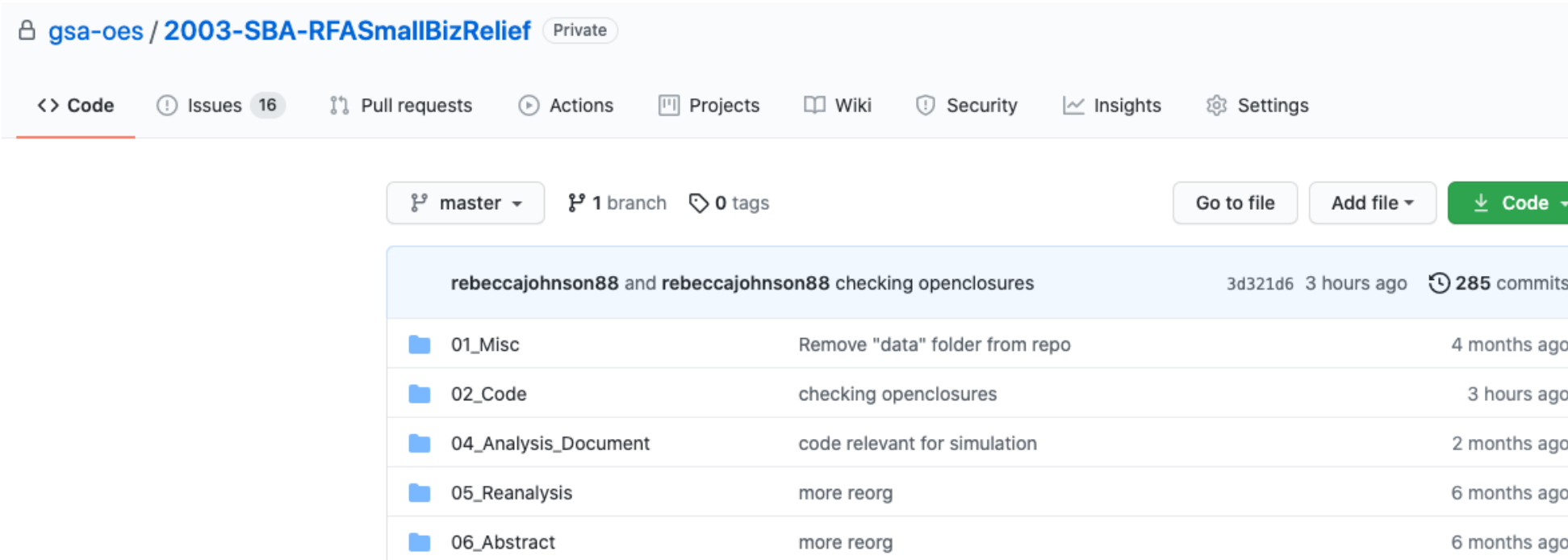
What is Git?

- Set of command line tools for version control (aka avoid finalfinal, finalrealthistime, etc.)
- "Distributed," or means that files/code, rather than only stored one place centrally, can be stored on all collaborators' machines

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



If you go to the url, get 404 error unless you're added as a collaborator: https://github.com/gsa-oes/2003-SBA-RFASmallBizRelief

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

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

```
317    ## rowbind the two
318  + all_rbind = rbind.data.frame(all, al
319  +             left_join(ylp %>% select(ye
320  +                                      de
321  +                     by = "yelp_id")
322  +
323  +
```

Example repo: public repo

Look familiar? https://github.com/jhaber-zz/QSS20_public/tree/main/activities

Ingredients of a repo: README

Should be more informative than the above example, e.g.:

README.md

PHA attributes: Section 8 study

Code related to spatial analysis for Sec 8 preferences study with Simone Zhang

Order to run

1. 0_loadPHApolygons_loadTractpolygons.Rmd

- Takes in:
  - Shapefiles from HUD's Estimated Housing Authority Service Area data: https://hudgis-hud.opendata.arcgis.com/datasets/estimated-housing-authority-service-areas
  - Tract shapefiles created by the script that uses `tigris` package to pull tracts for all state codes represented in the PHA data, and row 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 spatial overlap runtime)
  - Tests different overlap logics (intersect versus within) with one PHA and one state's tracts to build intuition
  - Tests plotting
- Outputs:
  - Two .RDS files, each containing `sf` format spatial polygon data for all US census tracts and all PHAs respectively
    a. tracts_foroverlap.RDS
    b. phas_foroverlap.RDS

2. 1_spatialmerge_loopcode.R

Ingredients of a repo: directories

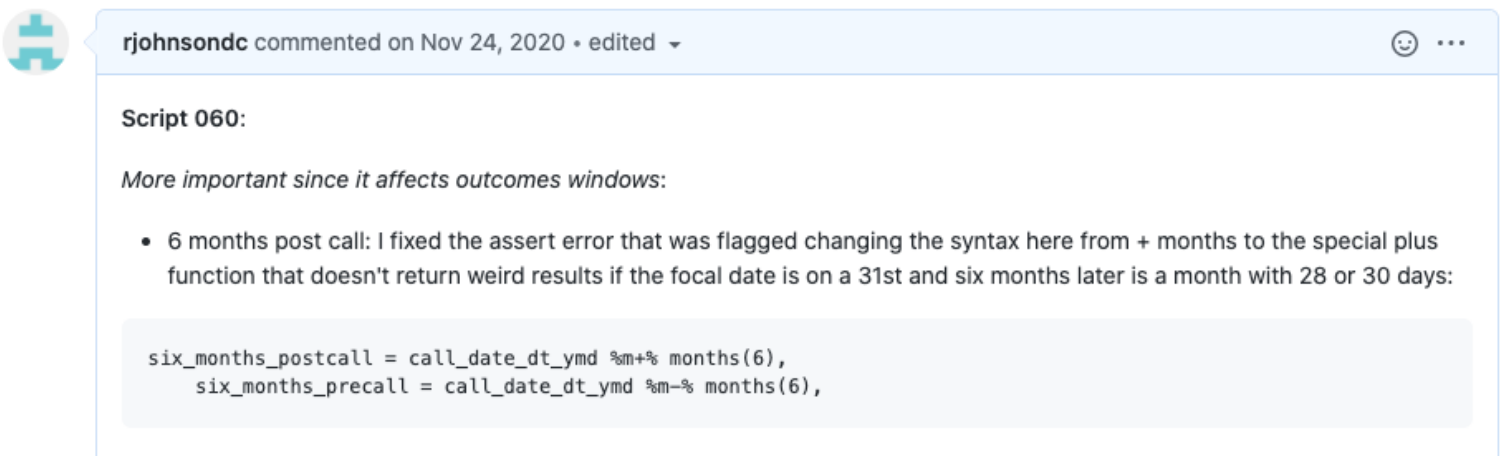
Command line syntax in previous slide is useful for org/reorg. For our class, we'll generally have two directories:

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)

Create a new repository: instructions

- On GitHub.com: Click the green "new" button
- Enter a name (for command line reasons, avoid spaces)
- Give a brief description
- Initialize with a readme
- Add a .gitignore (basically residual files you don't want in repo)
- Select a license

Contribute to a repository

1. Clone repo
2. Edit files
3. Send *all* changes to GitHub (use with caution!!)

```
git status
git add -A
git commit -m "this is what i changed"
git push
```

4. Send *specific file* changes to Github (more common)

```
git status
git add specificfile1.ipynb specificfile2.csv
git commit -m "this is what i changed"
git push
```

5. 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
```

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 the blank problem set (`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 using the website and instructions above; name it `qss20_f22_assignments`; add me (jhaber-zz) and Eunice (the TA; eunice30718) as collaborators
2. Clone the repo locally using your terminal/terminal emulator
3. Create a `code/` subdirectory
4. Create a `output/` subdirectory
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 to the `code` subdirectory
8. Push the changes to the `output` subdirectory
9. Using the GitHub website, edit the README to link to those changes
10. Assign Prof. Haber an issue
11. Make another change to a file locally (e.g., could edit the text file or add a comment to the code file) and try pushing. You should receive an error if you edited the README non-locally (i.e., on GitHub). Try to diagnose by googling, fix, and re-push.

For that last step...

Problem set two 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")
- Then, when you're ready for it to be graded, push two files to your repo:
  - The raw .ipynb
  - The compiled .html
- When you're done and ready for it be graded, assign me & TA a

Additional GitHub topics we may cover in a future session

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

Where we are

- Final projects:
  - Presentation by SIP lead Ashley Doolittle, Q&A
  - "Project shopping
- Piazza hello
- Basic command line syntax
- Git/GitHub
- **User-defined functions: lecture & activity from previous lecture**