

PPOL564: Data Science I

Unit 04: Workflow tools

Goals for today's session

- ▶ **Feedback discussion**
- ▶ Basic command line syntax
- ▶ Git/GitHub
- ▶ Working with LaTeX/Overleaf

Where we are

- ▶ Feedback discussion
- ▶ **Basic command line syntax**
- ▶ Git/GitHub
- ▶ Working with LaTeX/Overleaf

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 from class repo**
- ▶ **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?

- ▶ Mac default one- open up spotlight and search for terminal
- ▶ Windows built in console or terminal emulators - see list here https://rebeccajohnson88.github.io/PP0L564_datascience1_fall22/docs/software_setup.html

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 `ppol564_activities`
4. Within that folder, make another subfolder called `blank_activities`
5. Enter that subfolder and list its contents (should be empty)
6. Navigate back up to `ppol564_activities` 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/rebeccajohnson/Desktop/ppol564_activities/`
5. Move a file “down” a level in a directory
`mv copy_examplefile.txt blank_activities/`
6. Move a file “up” one level
`mv copy_examplefile.txt ../`
7. Up two levels:
`cd ../../`

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, if we're in the outer directory, deleting all files besides the ppol564 one in this dir

```
(base) rebeccajohnson@Rebeccas-MacBook-Pro-2 blank_activities % ls -tr  
ppol564.txt      ppol560.txt      ppol561.txt
```

```
find blank_activities/ -name 'ppol56[0|1].txt' -delete
```

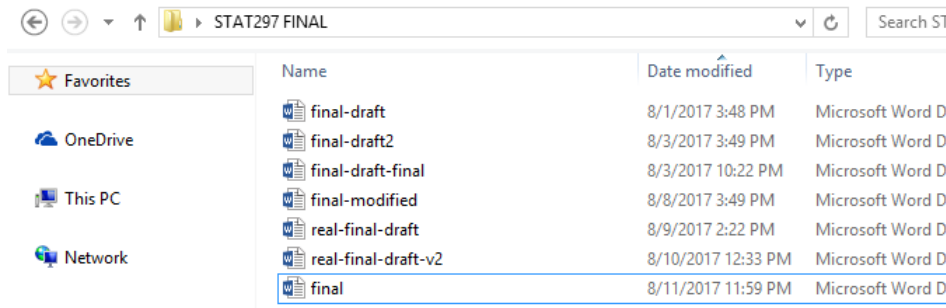
Activity (on your terminal/terminal emulator)

1. Delete the `blank_activities` directory in `ppol564_activities`
2. Use `touch` to create the following two files in the main `ppol564_activities`: `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

- ▶ Feedback discussion
- ▶ Basic command line syntax
- ▶ **Git/GitHub**
- ▶ Working with LaTeX/Overleaf

Motivation for Git/GitHub



Source: SMAC group

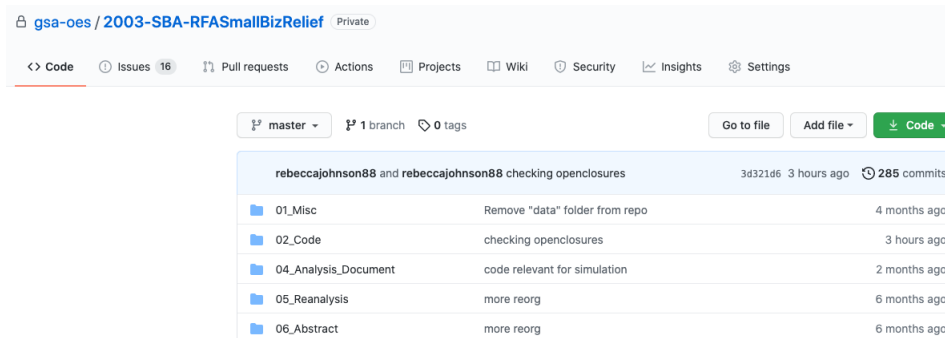
What is Git?

- ▶ Set of command line tools for version control (aka avoid finalfinal, finalrealthis time, 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



gsa-oes / 2003-SBA-RFASmallBizRelief Private

<> Code 1 Issues 16 Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

Go to file Add file Code

rebeccajohnson88 and rebeccajohnson88 checking openclosures		3d321d6 3 hours ago	285 commits
01_Misc	Remove "data" folder from repo	4 months ago	
02_Code	checking openclosures	3 hours ago	
04_Analysis_Document	code relevant for simulation	2 months ago	
05_Reanalysis	more reorg	6 months ago	
06_Abstract	more reorg	6 months ago	

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, all_alwaysclosed_wclosed)  
319 +     left_join(ylp %>% select(yelp_id, de  
320 +     by = "yelp_id")  
321 +  
322 +  
323 +
```

318



Example repo: public repo

Look familiar? `https:`

`//github.com/rebeccajohnson88/PP0L564_slides_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

- [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
 - `tracts_foroverlap.RDS`
 - `phas_foroverlap.RDS`
- [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, and (2) sensitive data should generally not be put in a repo, even if the repo is private (instead, read 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



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: new
- ▶ Enter a name (for command line reasons, avoid spaces)
- ▶ Give a brief description
- ▶ Initialize with a readme
- ▶ Add a .gitignore (basically residual files you dont want in repo)
- ▶ Select a license

Contribute to a repository

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

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

4. Send changes to Github (specific files)

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

5. Send changes to GitHub (files of a given type; eg you created a bunch of figures that you want to push)

```
git status
git add *.png
git commit *.png -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
```

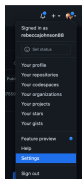

Activity 1: clone the public class repo so you can get recent changes

1. Open up terminal
2. Type:

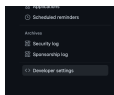
```
git clone https://github.com/rebeccajohnson88/PPOL564_slides_activities.git
```
3. Use `cd` to navigate to `activities/fall_22`
4. Open up a notebook and try editing an activity

Before next activity, you may need to create a personal access token on GitHub

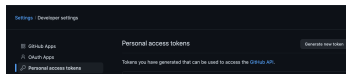
1. Settings on right hand side under your profile



2. Scroll down to "Developer settings" on left hand side



3. Click on personal access token and generate a personal access token



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

1. Create a new private repo using the website and instructions on slide 22; name it `ppol564_f22_assignments`; add me (`rebeccajohnson88`) as a collaborator
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, use `touch` to create a 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 include a link to the code directory (can just copy/paste the url)
10. Assign me an issue

Dealing with merge conflicts

With the last activity, let's see what happens if we try this:

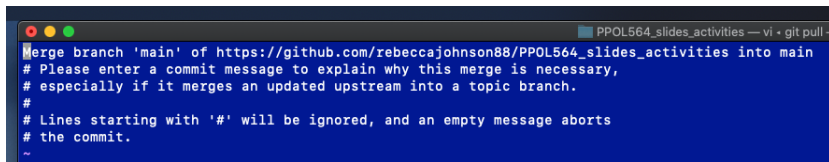
- ▶ Make another change to a file locally (e.g., could edit the text file or add a comment to the code file) and try pushing. You should receive an error if you edited the README non-locally.

Likely output:

```
(base) rebeccajohnson@rebeccas-macbook-pro-2 PPOL564_slides_activities % git push
To https://github.com/rebeccajohnson88/PPOL564_slides_activities.git
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/rebeccajohnson88/PPOL564_slides_activities.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Solution

- ▶ Type `git pull` to pull in remote changes. Two possible outcomes:
 1. **There are no merge conflicts:** you'll just get a message noting what changed and then you can proceed with `git push`
 2. **There are merge conflicts:** you'll get something that looks like this

A screenshot of a terminal window with a dark background. The title bar at the top shows three window control buttons (red, yellow, green) on the left and the text 'PPOL564_slides_activities — vi • git pull' on the right. The terminal content is white text on a dark blue background, showing a standard git merge message. The text is: 'Merge branch 'main' of https://github.com/rebeccajohnson88/PPOL564_slides_activities into main', followed by instructions to enter a commit message, and a note that lines starting with '#' are ignored. The cursor is at the end of the first line.

```
Merge branch 'main' of https://github.com/rebeccajohnson88/PPOL564_slides_activities into main
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
```

- ▶ **How to proceed depends on the text editor attached to your terminal:**
 - ▶ **Mac:** uses `vi` or `vim`. Commands are, with the `:` important:
 1. `:i` to insert text
 2. Escape key + `:wq` to write and quit
 - ▶ **Other common ones:** Emacs [discussion here of commands](#)
- ▶ **Can change:** <https://docs.github.com/en/get-started/getting-started-with-git/associating-text-editors-with-git>

Lame meme



Activity 3: move today's in-class activity over into your personal repo

- ▶ After cloning the class repo, you should have this file cloned locally within the `PP0L564_slides_activities` clone:

```
activities/fall_22/04_latex_output_examples_blank.ipynb
```

- ▶ Use the `mv` command to move that activity into the code folder of your personal repo

Additional GitHub topics we can review in office hours and/or future class

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

Where we are

- ▶ Feedback discussion survey
- ▶ Basic command line syntax
- ▶ Git/GitHub
- ▶ **Working with LaTeX/Overleaf**

Overview before activity

- ▶ LaTeX: typesetting language
- ▶ Can work with locally using things like TexMaker, etc.
- ▶ Here, we'll be interacting with it via Overleaf, which is similar to Google docs but for LaTeX and facilitates collaboration/easy(or easier...) troubleshooting of compile errors

Companies

TEAMS

Stack Overflow for Teams – Collaborate and share knowledge

▲

105

▼

🔖

I really want to convince my friends and family that LaTeX is the choice for them when it comes to formatting and creating beautiful documents. I am aware of the major advantages that come with using LaTeX but some are not convinced. Can someone please provide a side by side comparison of a Word document (or something of the sort) and a LaTeX document that shows the obvious and subtle differences between the two? I want people to look at it and say "Ahhh, I see it, there's a major difference".

Non-exhaustive list of things that can cause compilation errors

- Underscores or certain special characteristics without an “escape” before them– eg:

```
## causes error due to underscore without escape
The file is called: file_here.R
## works
The file is called: file\_here.R
## comments out rest of code after percent symbol
This increased by 5%
## works
This increased by 5\%
```

- Start entering math mode but fail to exit it, e.g.

```
## causes errors
We calculate fraction as  $\dfrac{5}{10}$  and then do...
## works
We calculate fraction as  $\dfrac{5}{10}$  and then do
```

“Environments”, or ways to go beyond standard text

► Itemized list

```
\begin{itemize}  
  \item First item...  
  \item  
\end{itemize}
```

► Numbered list

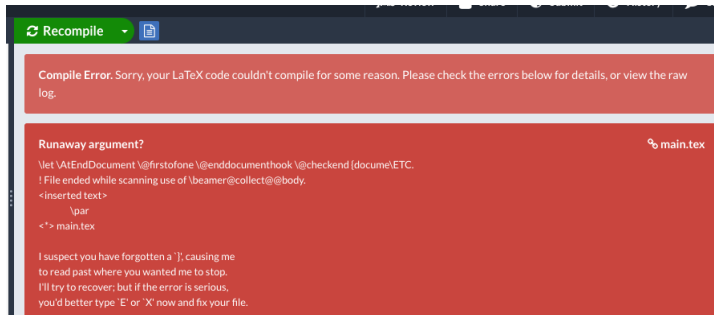
```
\begin{enumerate}  
  \item First item...  
  \item  
\end{enumerate}
```

► Figure

```
\begin{figure}  
  \caption{my caption}  
  \label{fig:myfig}  
  \includegraphics[scale = 0.5]{example_graphic.png}  
\end{figure}
```

Leads to another set of compilation errors

- ▶ Runaway argument or forgotten end group
- ▶ Usually means you began an environment but forgot to end it; can happen with long tables, deeply nested lists, etc. where easy to lose track



Example:

Compilation errors

- ▶ Common w/ complicated docs
- ▶ Ways to address beyond googling: try to recompile relatively frequently since especially on Overleaf, error messages are not always the most informative w.r.t. line numbers

Other useful commands

```
## create a numbered section and give it a label to cross-ref
\section{This is my section outlining disparities}
\label{sec:disparities}
```

```
## reference a section in text
In Section \ref{sec:disparities} I discuss...
```

```
## reference a table or fig in text
In Table \ref{tab:tabname}, I show why Figure \ref{fig:myfig} shows
```

```
## stop a table or figure from going into the next section
## (in addition to stuff at the start of the \begin{table} command
\FloatBarrier
```

Break for LaTeX tables and figures activity

- ▶ [Link to template to copy over](#)
- ▶ Link to Python activity:

[04_latex_output_examples_blank.ipynb](#)