Terminal, Git, and Jupyter

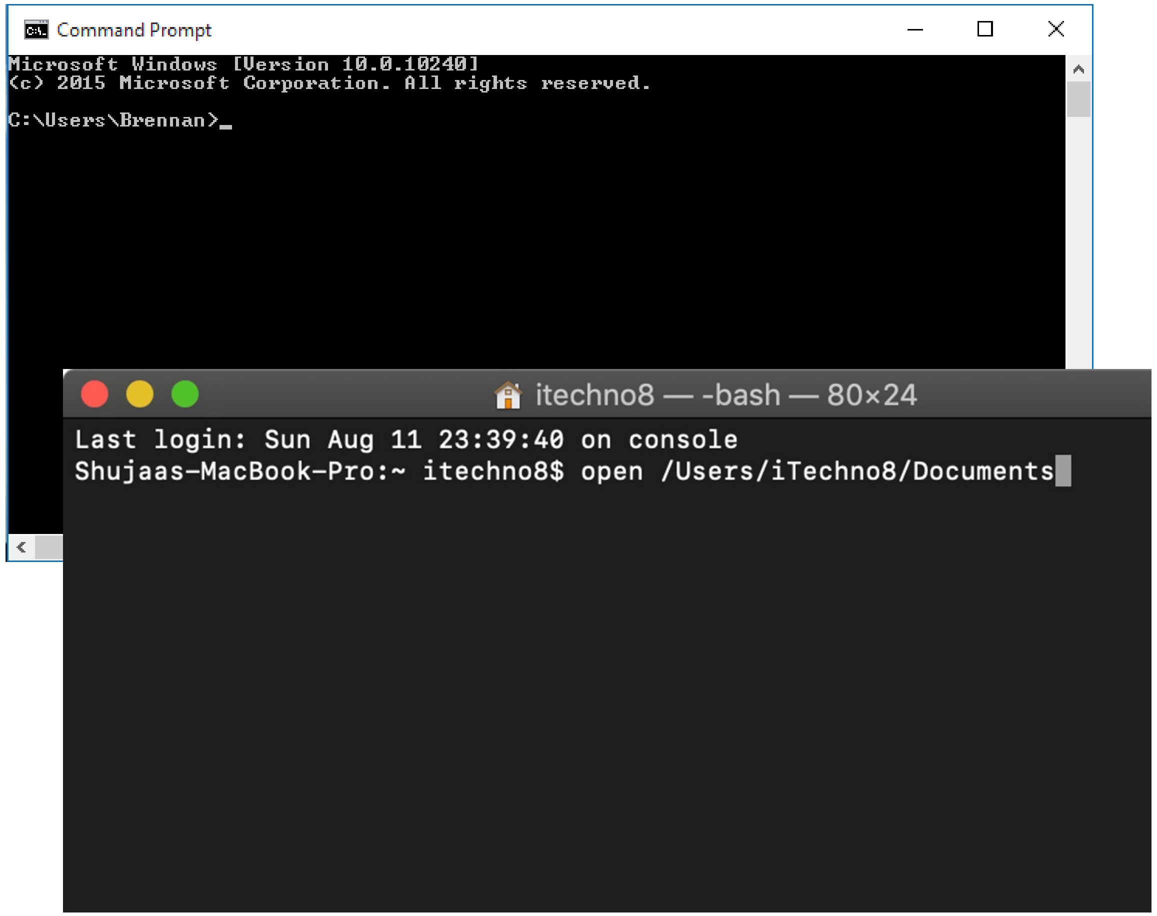
ACE 592 SAE

Outline

- What is the terminal?
- What is Git?
    - How to use Git
    - How to use GitHub
    - Git demonstration
- What is Jupyter?
    - Initiating a notebook.
    - Programming using the notebook.
    - Notebook and Binder demonstration

The Terminal

- The way to access low-level functions on your computer.
  - Copying, moving, running programs.
  - Done visually with a “Guided User Interface” (GUI)
- Why do we need it?
  - Less CPU.
  - Some things are MUCH easier this way.
  - Servers typically have no GUI.



```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Brennan>_
```

```
Last login: Sun Aug 11 23:39:40 on console
Shujaas-MacBook-Pro:~ itechno8$ open /Users/iTechno8/Documents
```

The Two Terminal “Languages”

Windows

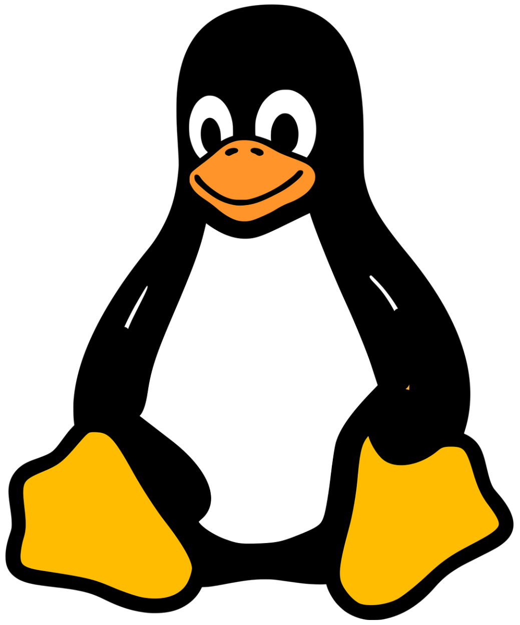
- Originated with DOS.
- Language is “cmd” and usually called “command line.”
- Used almost exclusively on Windows computers using “Command Prompt” application.

Unix (Mac OSX/Linux)

- Originated with Unix.
- Language is “bash” and usually called “shell scripting.”
- Used on Linux and Mac OSX, likely getting ported to Windows terminal soon.

The Open Source Overlord: Linus Torvalds

We’re only using bash, and here’s why:



- Computing clusters almost all use Linux OS (Ubuntu, Red Hat, etc.)
- To use these servers, you need to know bash, as there is no GUI available.
- By using Jupyter and Git first with bash, you will develop the skills to eventually use big servers for computations.

Bash Commands Demonstration:

- Navigating directories
  - “cd” – change directory.
  - “ls” – print directory contents.
- Move files
  - “rm” – delete file (really dangerous command!).
  - “mv” – move file.
  - “cp” – copy file.
- Manipulate files
  - “less” – Look at text file.
  - “mkdir” – create a directory.
  - “grep” – search for files.

Bash Demonstration

What we will use it for:

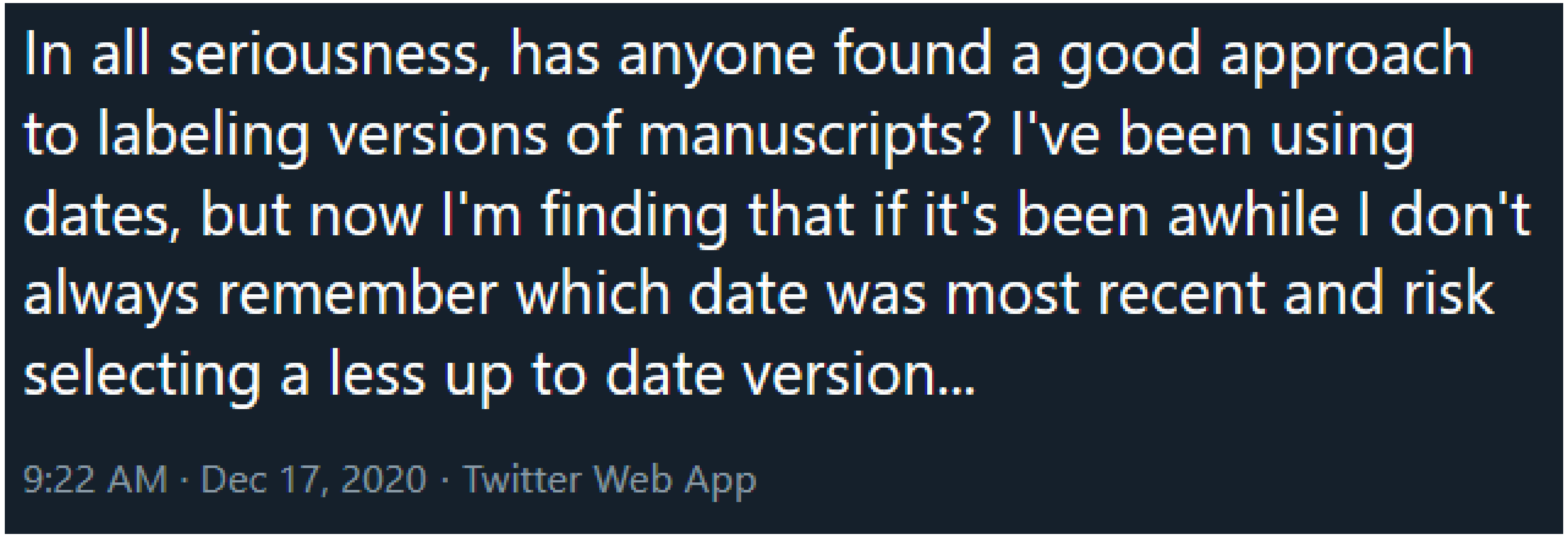
- Git version control.
- Downloading Python packages.
- Managing conda “environments.”
- Launching Jupyter notebooks.

All of these have GUI equivalents, but are much slower.

Also, you won’t have a GUI on a computing cluster!

Version Control with Git

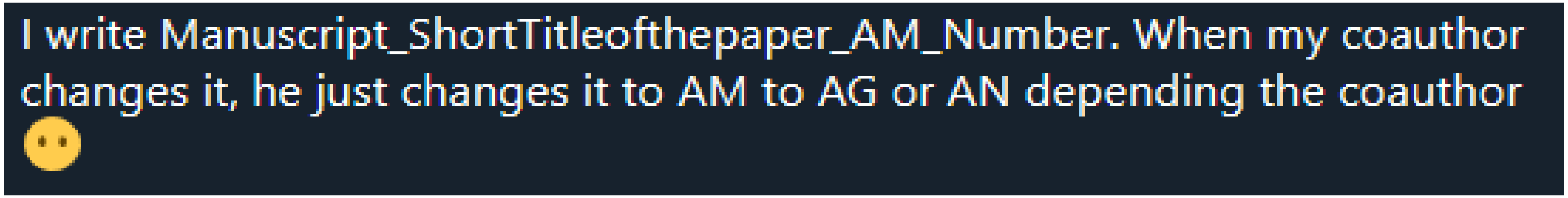
“Has this ever happened to you?”

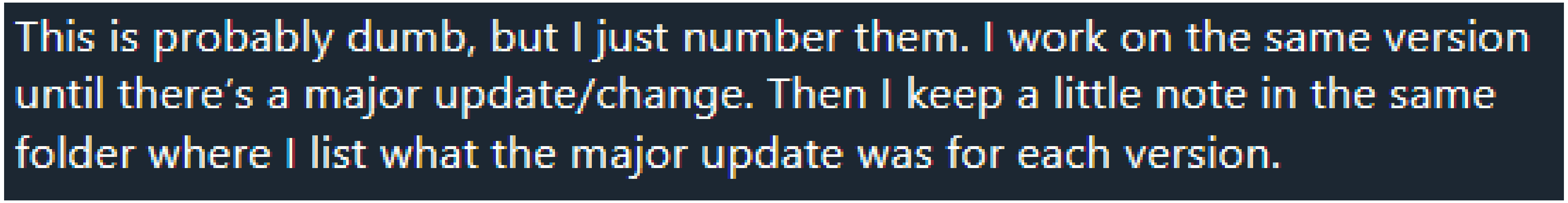


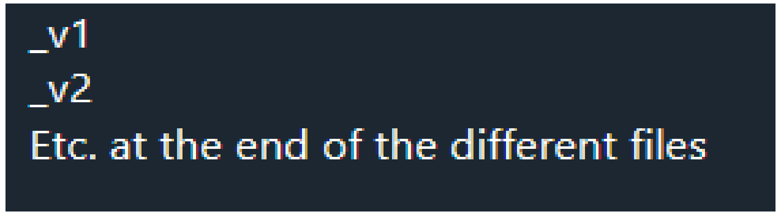
In all seriousness, has anyone found a good approach to labeling versions of manuscripts? I've been using dates, but now I'm finding that if it's been awhile I don't always remember which date was most recent and risk selecting a less up to date version...

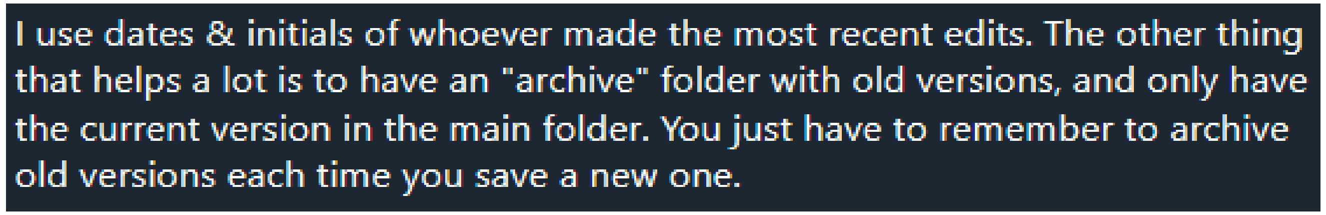
9:22 AM · Dec 17, 2020 · Twitter Web App

Some Common Approaches


I write Manuscript_ShortTitleofthepaper_AM_Number. When my coauthor changes it, he just changes it to AM to AG or AN depending the coauthor 😶


This is probably dumb, but I just number them. I work on the same version until there's a major update/change. Then I keep a little note in the same folder where I list what the major update was for each version.


_v1
_v2
Etc. at the end of the different files


I use dates & initials of whoever made the most recent edits. The other thing that helps a lot is to have an "archive" folder with old versions, and only have the current version in the main folder. You just have to remember to archive old versions each time you save a new one.

Issues with these approaches:

Trusting yourself to remember why “v1” and “v2” are different.

Making tons of files in your directory is confusing.

How is someone supposed to know your own system?

This approach doesn’t work with code.

What is Git?

- Git is an open-source, “version control” software.
  - Open-source = free to use and copy.
  - Version control = keeping a history of the versions of a file.
- Why it’s better:
  - Keeps one file name but keeps “snapshots” of the file and the history.
  - Write a message to yourself or your co-authors saying what changed.
  - Can retrace your steps to figure out what broke.

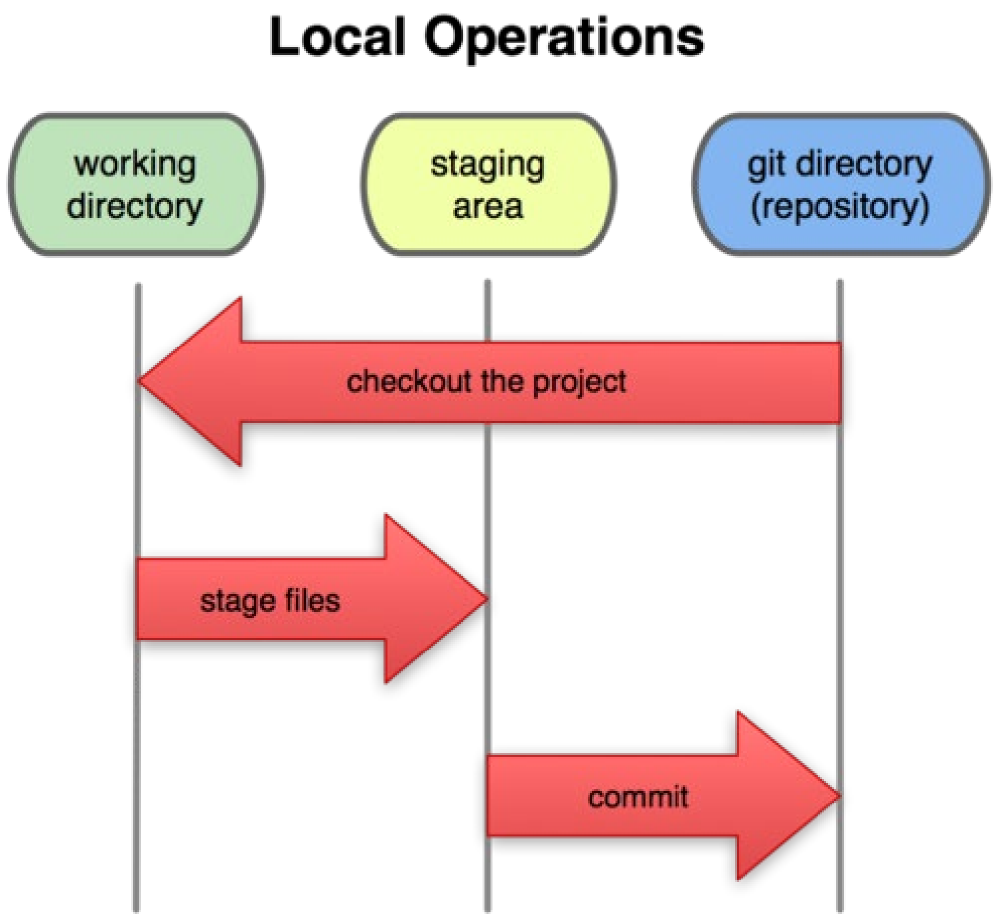
What does git do?

- Every time you make a change (a “commit”) to a file in the tracked directory (called a “repository”), it:
  - Makes a copy of that file (a “blob”).
  - Has a message describing the change.

- When you make another change, git can compare your previous version to your current version (“git diff”).

- Very easy to change the directory to a previous version (called a “revert”).

Making Changes in Git



- Your working directory is what you do while you're working.
- "git add" – this adds the files to the **staging** area.
- "git commit" – this locks in all the files in the staging area to the "**repository**" or directory.

Once the changes are committed, git takes a snapshot of **all of these changes** and saves this "version" of your directory.

Why do we need a staging area?

Why do we need a staging area?

- Imagine two code files: “analysis.do” and “data_clean.do”
- I see a problem with the data cleaning. I fix it by making a change to **both** files.
- After making the changes, I’ve realized it now doesn’t work.

Q: What happens if I revert only “analysis.do”?

Why do we need a staging area?

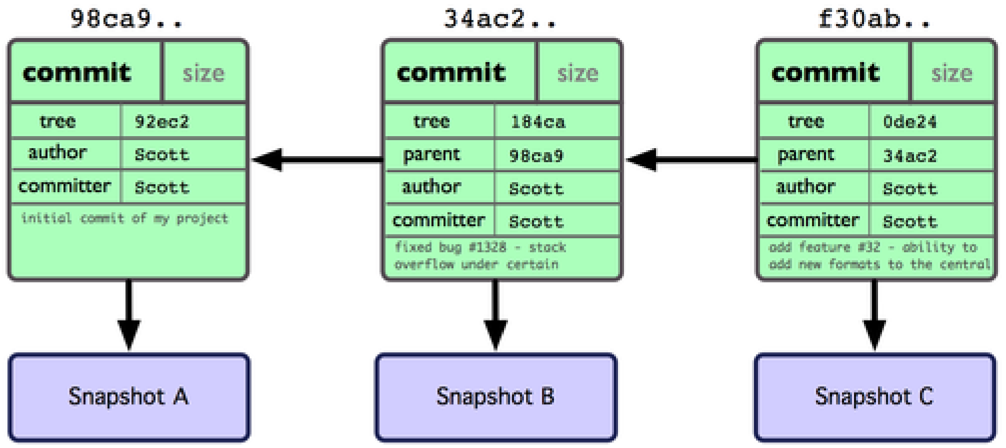
- Imagine two code files: “analysis.do” and “data_clean.do”
- I see a problem with the data cleaning. I fix it by making a change to **both** files.
- After making the changes, I’ve realized it now doesn’t work.

Q: What happens if I revert only “analysis.do”?

A: Probably still breaks.

Learning to Commit in Batches

- Git forces you to think in **“commits”** instead of individual files.
- Think in terms of the relationships between files.
- Which **“version”** of your project do you want to save?



The Anatomy of a Common Git Command

command

file

git add analysis.do

“this is a git command”

“add a change to the staging area”

“add the changes to this file”

The Anatomy of a Common Git Command

	command	flag	message
git	commit	-m	“I changed stuff”
“this is a git command”	“commit my staged changes”	“use the following text as a message”	A commit message

A Typical Git Workflow

What you are doing	Git command
Made a change to analysis.do	
Compare your changes to the last commit.	git diff analysis.do
Staged the change to analysis.do	git add analysis.do
Made a change to data_clean.do	
Staged that change.	git add data_clean.do
Committed both the changes into one batch and wrote a message explaining what I did.	git commit -m “Fixed error in data cleaning that caused analysis to fail.”

What if my changes break everything?

If your changes **are not committed**:

“git checkout” returns **the file** to whatever it was at the last commit.

If your changes **are committed**:

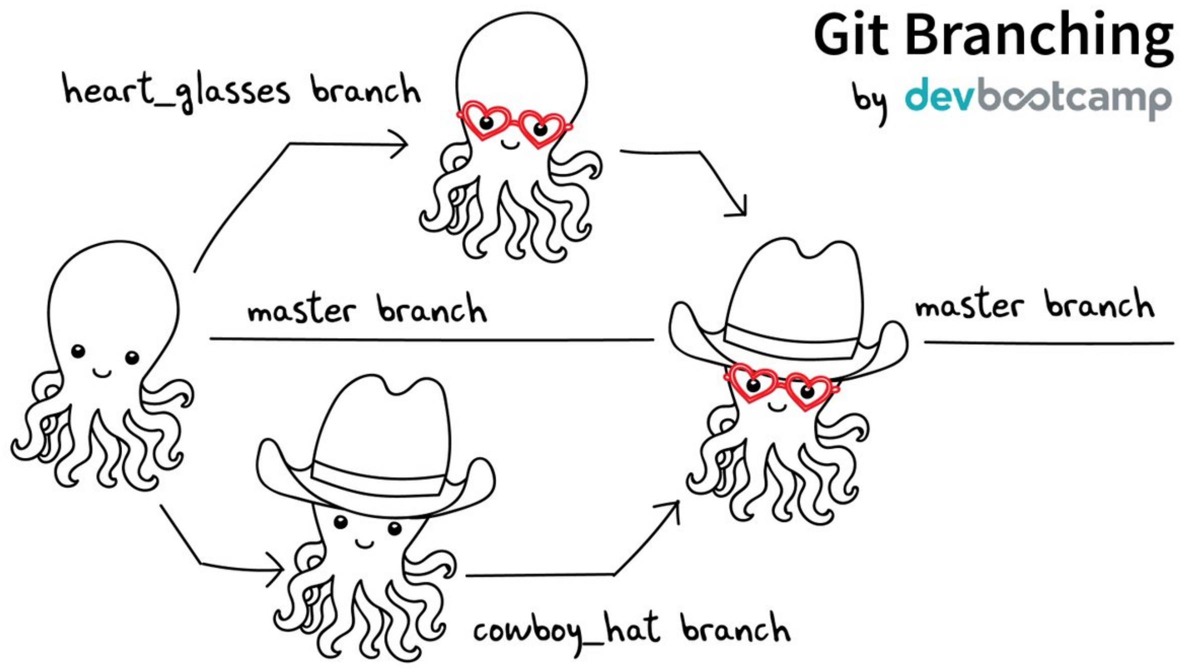
“git revert” returns **the whole directory** to a previous commit

Since it keeps a history, **you will never lose the previous versions.**

Git stops you from destroying your own code!

Git Branches

- Another way to stop yourself from breaking stuff.
- Your main branch is "master" but you can create a **parallel version** of "master" to make significant changes that won't affect "master."
- If the changes are good, "git merge" combines the branches.
- If there are branch conflicts, git will make you sort them out.



What should I track?

Git is good for:

- Code (.py, .do, .m, .r, .ipynb)
- Text files (.txt, .tex)
- Small (<10mb) data files (.csv, .json)

Git is not good for:

- Big data files.

For the files you don't want tracked, make entries in your ".gitignore" file

So what is GitHub?

- A place to store your repository on the internet.
- Why would you want to do this?
  - You work across several machines.
  - You have collaborators.
  - You want to share your code with others.

- GitHub is now an engine for creating open-source software and a place to find and share code.

More terminology

“**local**” = your computer.

“**remote**” = your GitHub repo.

“**pull**” = grab changes from a remote.

“**push**” = make changes to a remote.

The Anatomy of a Common Git Command

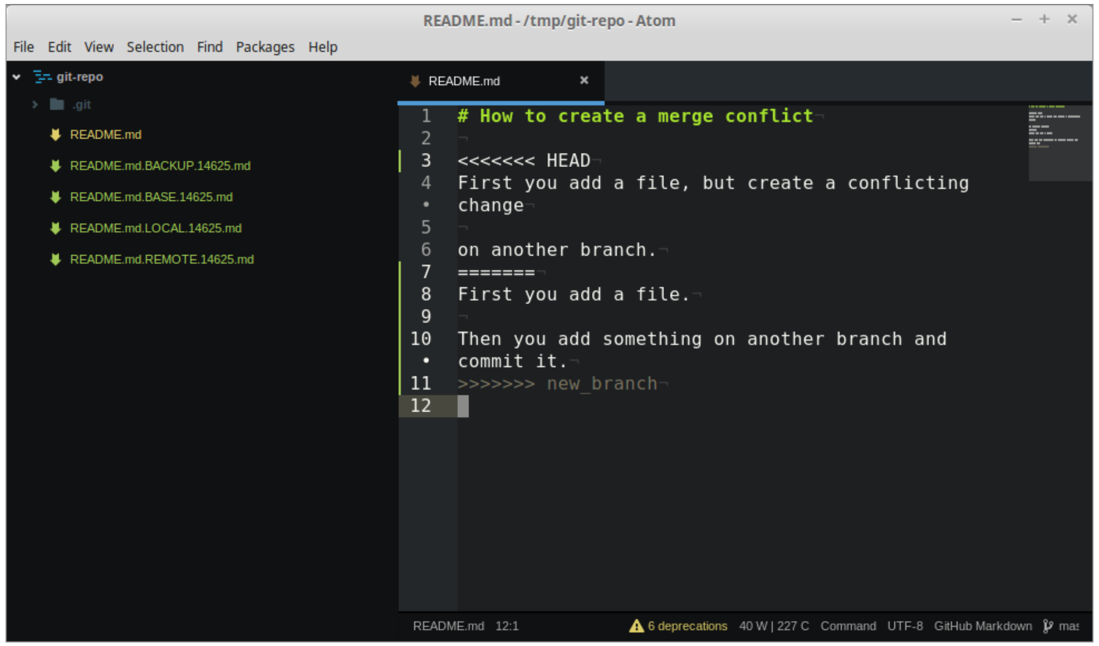
	command	remote	branch
git	push	origin	master
“this is a git command”	“push my changes”	“get it from the remote called origin”	“pull the master branch”

A typical git + GitHub workflow

What you are doing	The git command
Update my repo on this computer with whatever I did to the repo on GitHub.	git pull origin master
Do some stuff	git add “analysis.do”
Tell git what you did	git commit –m “I made brilliant amazing changes because I am amazing”
Now update your GitHub remote.	git push origin master

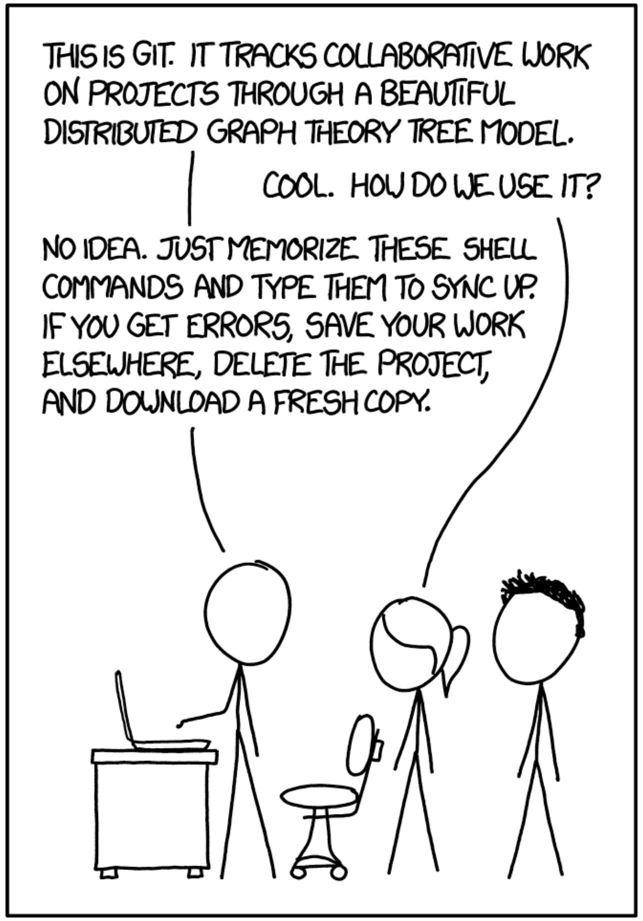
Conflict Resolution

- What if your GitHub “analysis.do” is different than the one on your local?
- This is a “merge conflict” and git will make you fix it.
- It will edit your file to look like the one on the right:
- HEAD = your local changes.


README.md - /tmp/git-repo - Atom
File Edit View Selection Find Packages Help
git-repo
.git
README.md
README.md.BACKUP.14625.md
README.md.BASE.14625.md
README.md.LOCAL.14625.md
README.md.REMOTE.14625.md
README.md
README.md 12:1
6 deprecations 40 W | 227 C Command UTF-8 GitHub Markdown mas

```
# How to create a merge conflict

<<<<<<< HEAD
First you add a file, but create a conflicting
change

on another branch.
=======
First you add a file.

Then you add something on another branch and
commit it.
>>>>>>> new_branch
```

Git Demonstration


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Your Next Task:

1. Get a GitHub account
2. Install git on your computer.
3. Clone the ACE592 repository.

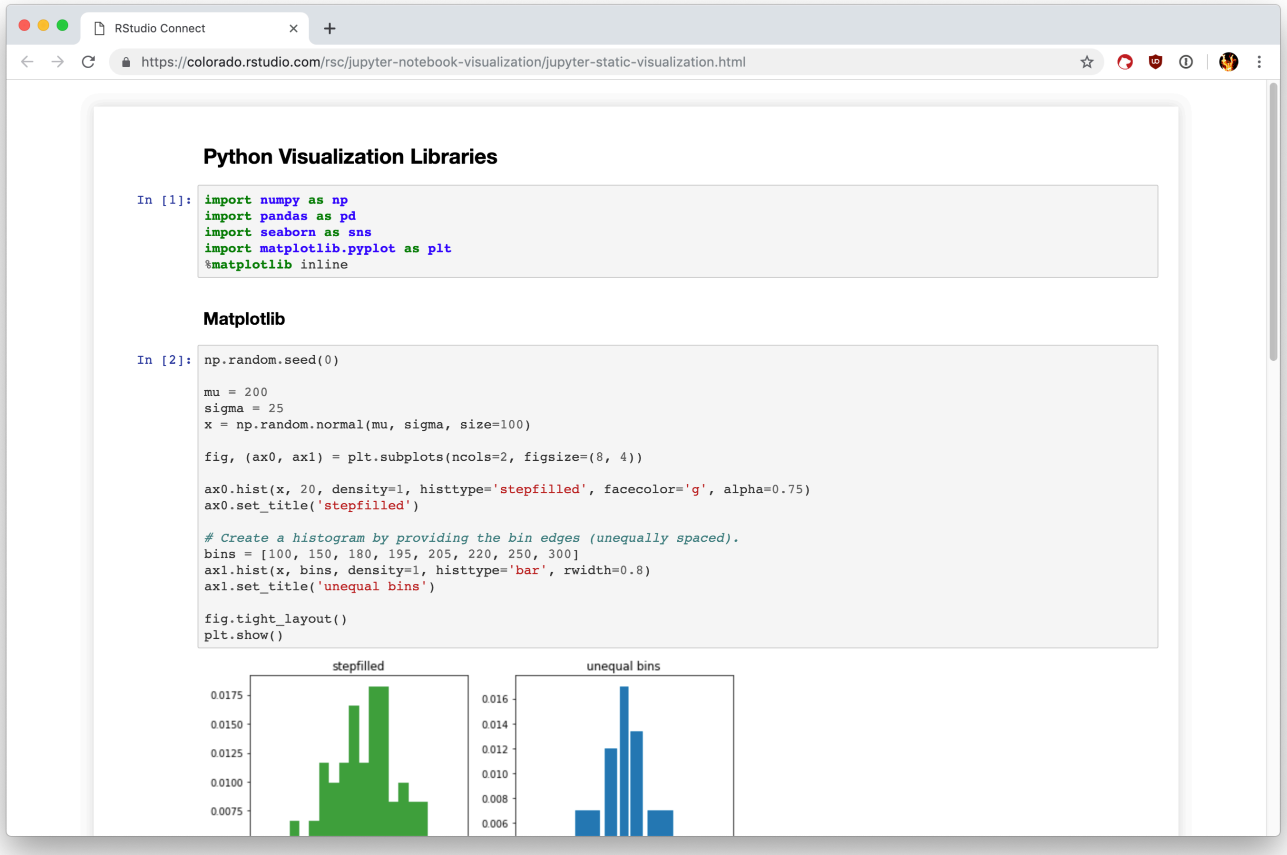
Jupyter Notebooks

IDE Architect:

Fernando Perez

Why Jupyter Notebooks?

- A good way to interactively code.
- Good support for both R and Python (also STATA).
- Makes mark ups really easy.
- Makes your code interactive.


RStudio Connect
https://colorado.rstudio.com/rsc/jupyter-notebook-visualization/jupyter-static-visualization.html
stepfilled
unequal bins
0.0175
0.0150
0.0125
0.0100
0.0075
0.016
0.014
0.012
0.010
0.008
0.006

Python Visualization Libraries

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
```

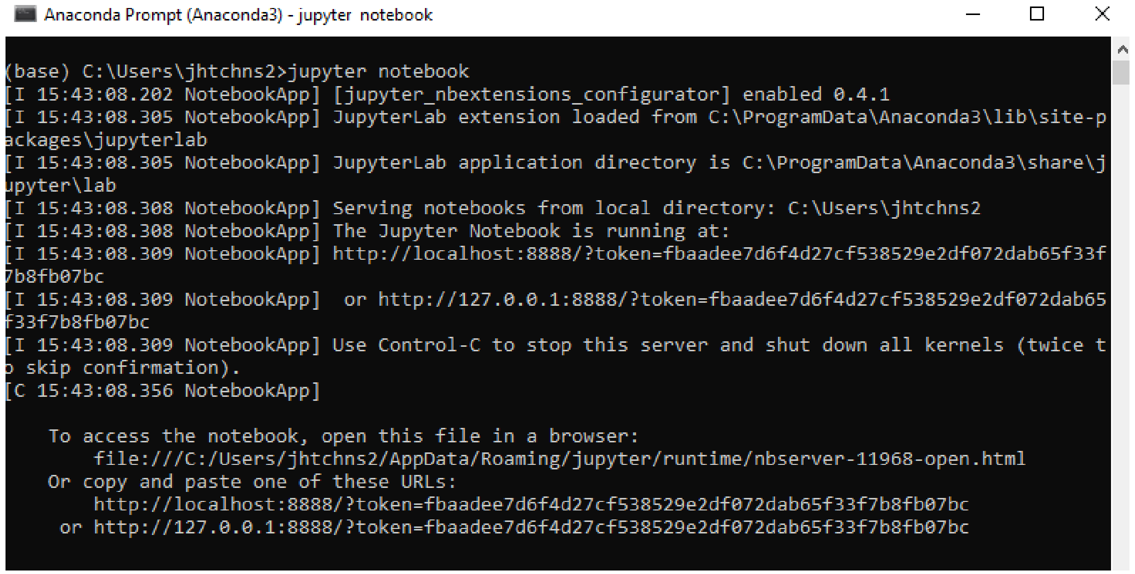
Matplotlib

```
In [2]: np.random.seed(0)

        mu = 200
        sigma = 25
        x = np.random.normal(mu, sigma, size=100)

        fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(8, 4))

        ax0.hist(x, 20, density=1, histtype='stepfilled', facecolor='g', alpha=0.75)
        ax0.set_title('stepfilled')

        # Create a histogram by providing the bin edges (unequally spaced).
        bins = [100, 150, 180, 195, 205, 220, 250, 300]
        ax1.hist(x, bins, density=1, histtype='bar', rwidth=0.8)
        ax1.set_title('unequal bins')

        fig.tight_layout()
        plt.show()
```

How to Start a Notebook

Terminal

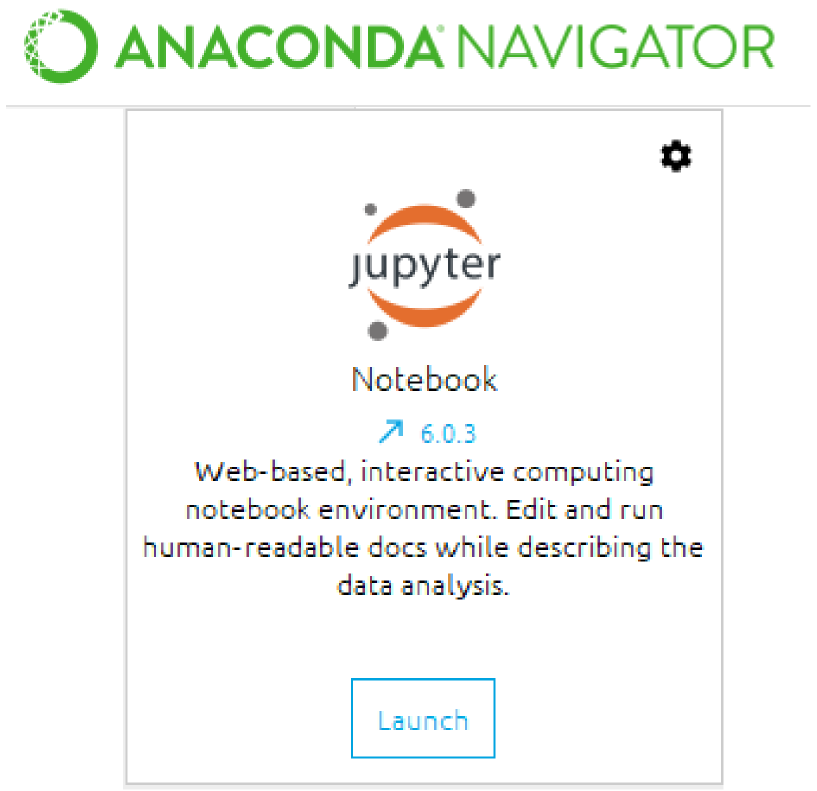
- Start a terminal and type “jupyter notebook”
- Keep terminal window open.



Anaconda Prompt (Anaconda3) - jupyter notebook

```
(base) C:\Users\jhtchns2>jupyter notebook
[I 15:43:08.202 NotebookApp] [jupyter_nbextensions_configurator] enabled 0.4.1
[I 15:43:08.305 NotebookApp] JupyterLab extension loaded from C:\ProgramData\Anaconda3\lib\site-p
ackages\jupyterlab
[I 15:43:08.305 NotebookApp] JupyterLab application directory is C:\ProgramData\Anaconda3\share\j
upyter\lab
[I 15:43:08.308 NotebookApp] Serving notebooks from local directory: C:\Users\jhtchns2
[I 15:43:08.308 NotebookApp] The Jupyter Notebook is running at:
[I 15:43:08.309 NotebookApp] http://localhost:8888/?token=fbaadee7d6f4d27cf538529e2df072dab65f33f
7b8fb07bc
[I 15:43:08.309 NotebookApp]  or http://127.0.0.1:8888/?token=fbaadee7d6f4d27cf538529e2df072dab65
f33f7b8fb07bc
[I 15:43:08.309 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice t
o skip confirmation).
[C 15:43:08.356 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/jhtchns2/AppData/Roaming/jupyter/runtime/nbserver-11968-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=fbaadee7d6f4d27cf538529e2df072dab65f33f7b8fb07bc
     or http://127.0.0.1:8888/?token=fbaadee7d6f4d27cf538529e2df072dab65f33f7b8fb07bc
```

GUI



Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Comparing the Two Methods

Terminal

- **Pros**: fast, can specify port number, can use all the options.
- **Cons**: less intuitive, have to keep terminal windows open.

GUI

- **Pros**: don't have to keep terminal window open.
- **Cons**: slow, hard to specify options.

For this class, you can start with the GUI but eventually you need to transition to doing it from the Terminal.

- **Faster.**
- **Specifies more options.**
- **Works on a server**

Jupyter Notebook Demonstration