The Command Line Interface, Conda and Git

Command Line Interface

- Nowadays, you almost certainly mostly interact with a computer using graphical windows and drop down menus. This is called a Graphical User Interface or GUI.
- Before GUIs people used to interact with a computer by typing text and reading text printed by the computer. This is called a Command Line Interface or CLI
- All computers still allow you to use a CLI and there are many reasons why one might want to do so.

Command Line Interface Advantages

- It is possible to know hundreds or thousands of commands, many more than you could possibly have on a drop down menu.
- If you connect to a machine remotely it can be difficult to set up a GUI and often is much easier to use a command line interface.
- When people write code and aren't supported by giant corporations they usually don't have time to write GUIs (scientist are an excellent example of this).
- Text commands can be scripted and run inside computer code.

Terminal, Anaconda Prompt

- To use a command line interface one has to run a program with a command line.
  - Mac OS X - default is Terminal, default shell is zsh, runs linux commands.
  - Windows - default is PowerShell but this is just Dos commands. If you want to run python from your anaconda distribution you will need to run Anaconda Prompt. You can run a shell to run a linux command shell, but it might take some work.
- You can also access a command line in VS Code and Jupyter notebook.

FILES

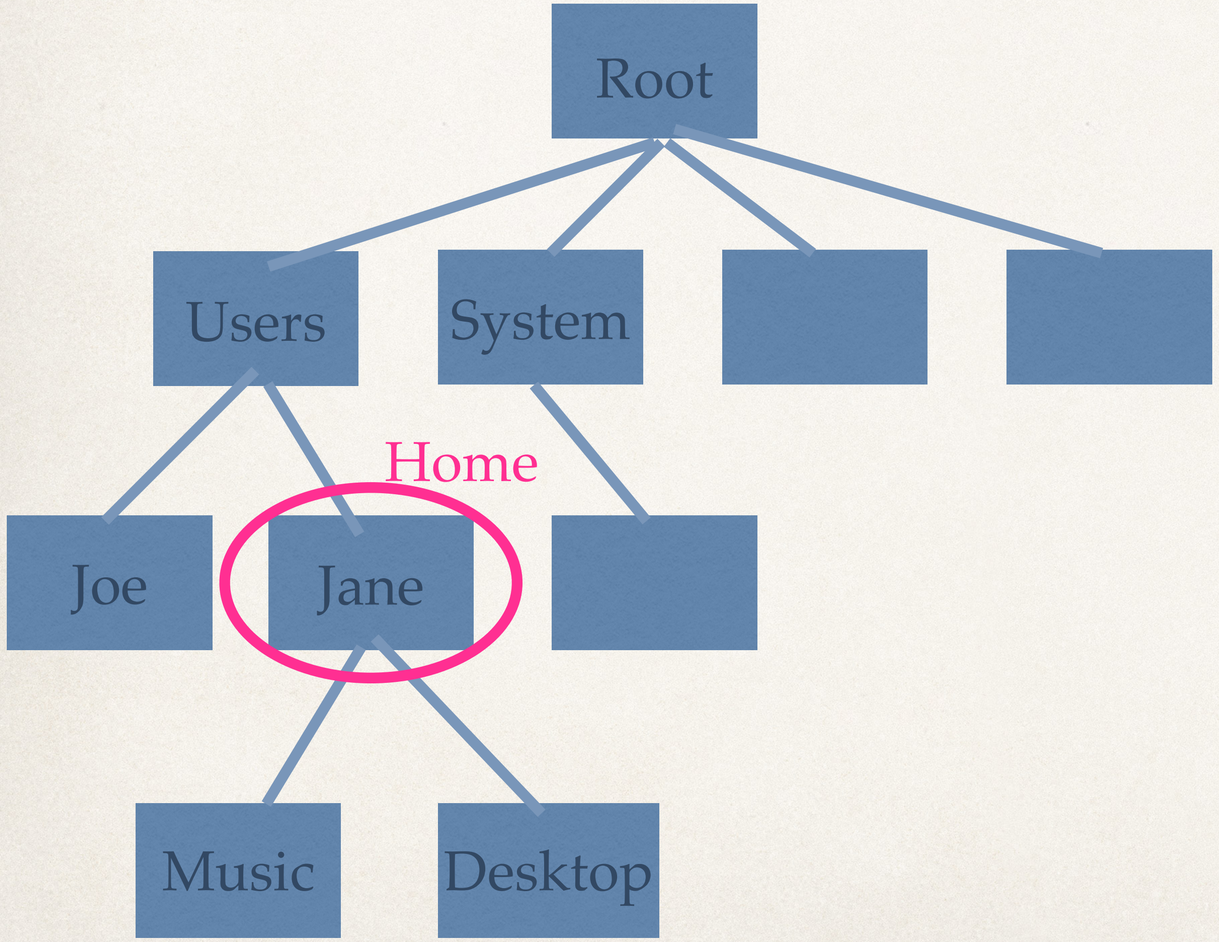
- One of the main uses of a computer is storing and retrieving data. While the data itself is just stored as zeros and ones, the filesystem is how this information is organized so that is easier for you and software to find.

- In terms that are antiquated now, each chunk of information is called a file, files can be grouped in a directory (also called a folder) and the filesystem refers to the overall structure of all the folders and directories.

- A file can be of many different types; a music file, an image, a video, code, or a text file. Any information stored on a computer is a type of file. Many files have an extension, usually the last 2 to 4 letters after a period that indicate what type of file it is.

- Some examples; .doc, .pdf, .mp3, .mp4, .jpg, .gif, .log, .txt, .mov, .csv. Note that these extensions are meant to be helpful, they don't change the file type, and aren't needed, though some programs will have difficulty reading a file without the right extension.

File System



- Directories (or folders) are organized in a hierarchy.
- The top level is called root.
- Usually a person's 'Home' directory is a few levels down. This is where you are primarily supposed to put your personal files.
- You can only write files in this directory and sub directories if you don't have administrator status.

PATHS

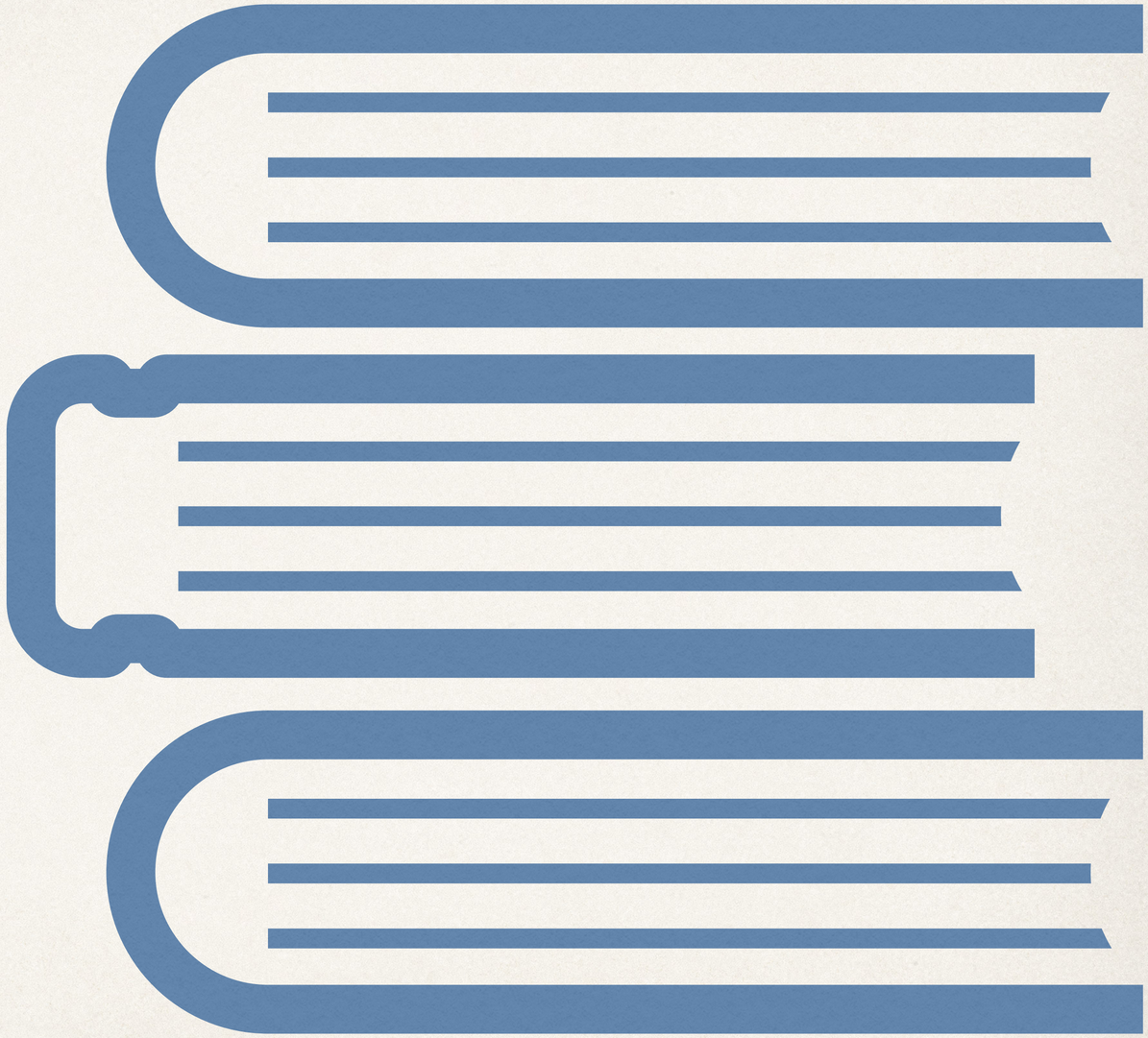
- On the command line the files system is represented by / (Unix) or \ (DOS) and root is just a /.
- /Users/Jane/Desktop would be the path to Jane's Desktop.
- On the command line you can see your current location with pwd (Unix) or cd (DOS).
- You can change your directory with cd <dirname> or cd .. (goes up 1).
- You can get a list of files and directories in your current directory with ls (Unix) or dir (DOS).
- Tab (Unix) will complete a name if there is only option or show options if there are more.

Unix Commands

- pwd - present working directory.
- cd <dirname> - change directory to dirname, .. for up one.
- ls - list directories and files in a directory
- cp <filename1> <filename2> - copy filename1 to filename2
- mv <filname1> <dirname> - move filename1 to directory dirname
- which <filename> - gives full path to filename
- ssh hostname - securely log into remote machine
- scp <filename> <hostname:path> - securely copy filename to remote machine directory

Exercise 1

Check your current directory and list the files in it. Then cd into a directory. Keep going into new directories until you can't go any farther. What is the path to your current location. Go all the way up to root and list the directories. Try to make it back your home directory in a single cd command.



Package Management

- Long ago people realized that keeping track of the programs on a computer and making sure that various programs worked together was tough. So they invented software to keep track of this for us.
- This is the Apple App Store or Microsoft Store for many programs. But mostly for big developers.
- Python packages can be managed with conda or pip. Since you all have an Anaconda installation I would suggest always trying conda first and using pip only if needed.
- Both codes will install packages and figure out if you have the right other packages installed for you. Also they will place the packages where your python installation can find them.

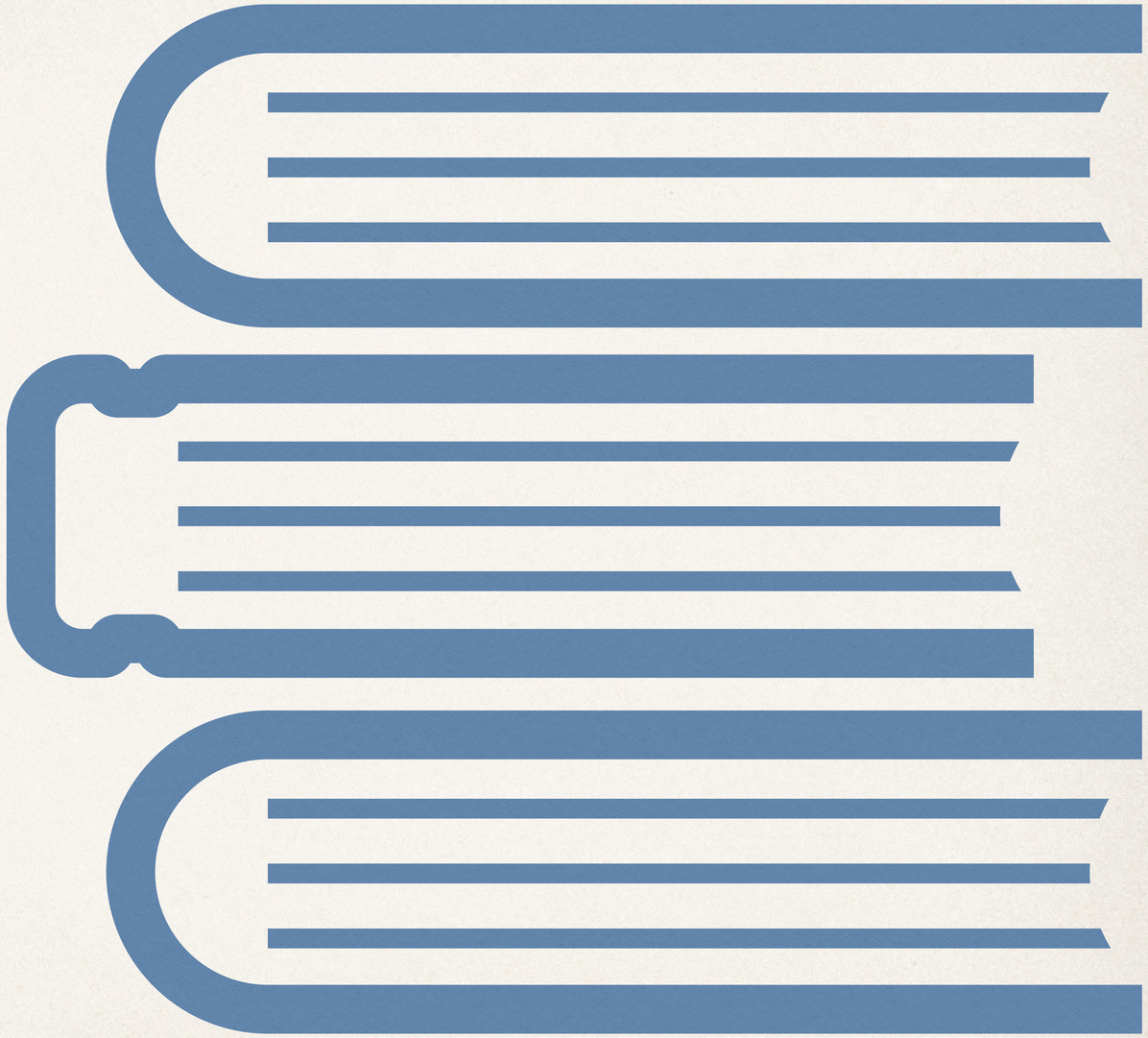
Conda

- Conda commands:
  - conda list - list all installed packages
  - conda list <name> - list installed packages that match name
  - conda search <name> - look for packages to install that match name
  - conda install <name> - install the package name

https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html

Exercise 2

Use conda to check that you have numpy, matplotlib, pandas, seaborn, astropy and plotly installed. Install any that you don’t have.



Version Control

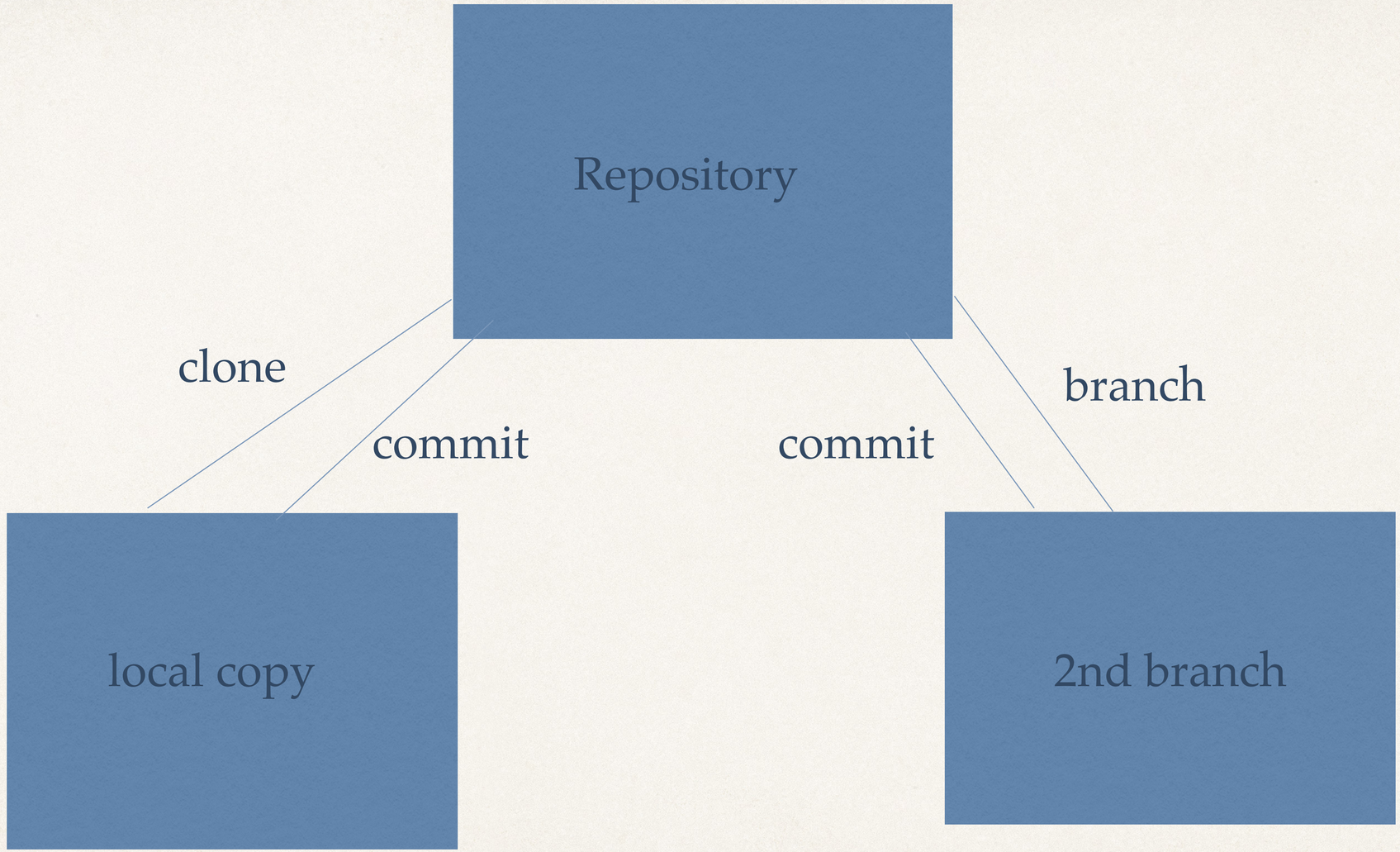
- A way to keep track of changes to your code.
- Also a way to collaborate on writing code with others.
- Many implementations though git is becoming the most common one.
- Also can be combined with a website to allow easy access to others, even people you don’t know.
- Widely used in open source programming.

Version Control

- Someday you will spend a long amount of time writing a fairly large and complicated code. When it runs and all the bugs have been worked out you will be very happy.
- Then you will want to make some minor revisions to your code, but after you do your code will no longer work.
- Not being able to figure out exactly what changes you made to the formally working version you will get very frustrated and wish you had saved a copy of the version that worked.
- THIS IS WHY WE USE VERSION CONTROL!

Version Control

- The basic idea of version control is that your files are kept in what is called a repository.
- However, you do not edit the files in the repository. Instead you check out a copy of them for you to work on.
- When you are done working on the files you commit them back to the repository.
- The version control software keeps track of each of these changes and allows you to check out an older copy if you want.
- It is also possible to create branches, where you change different things, or different people change different things and then merge them back at some later time.


Repository
clone
commit
commit
branch
local copy
2nd branch

git

- The most popular version control system is probably git, and that has a lot to do with GitHub a website that freely hosts your files as long as they are open source.
- Installing Anaconda Python should have installed git.
- git can also be installed from a package management system or with a GUI.
- You can either use the command line or a GUI, they do the same things. You should at least practice on the command line so you understand what the GUI is doing.
- check on terminal with “which git”

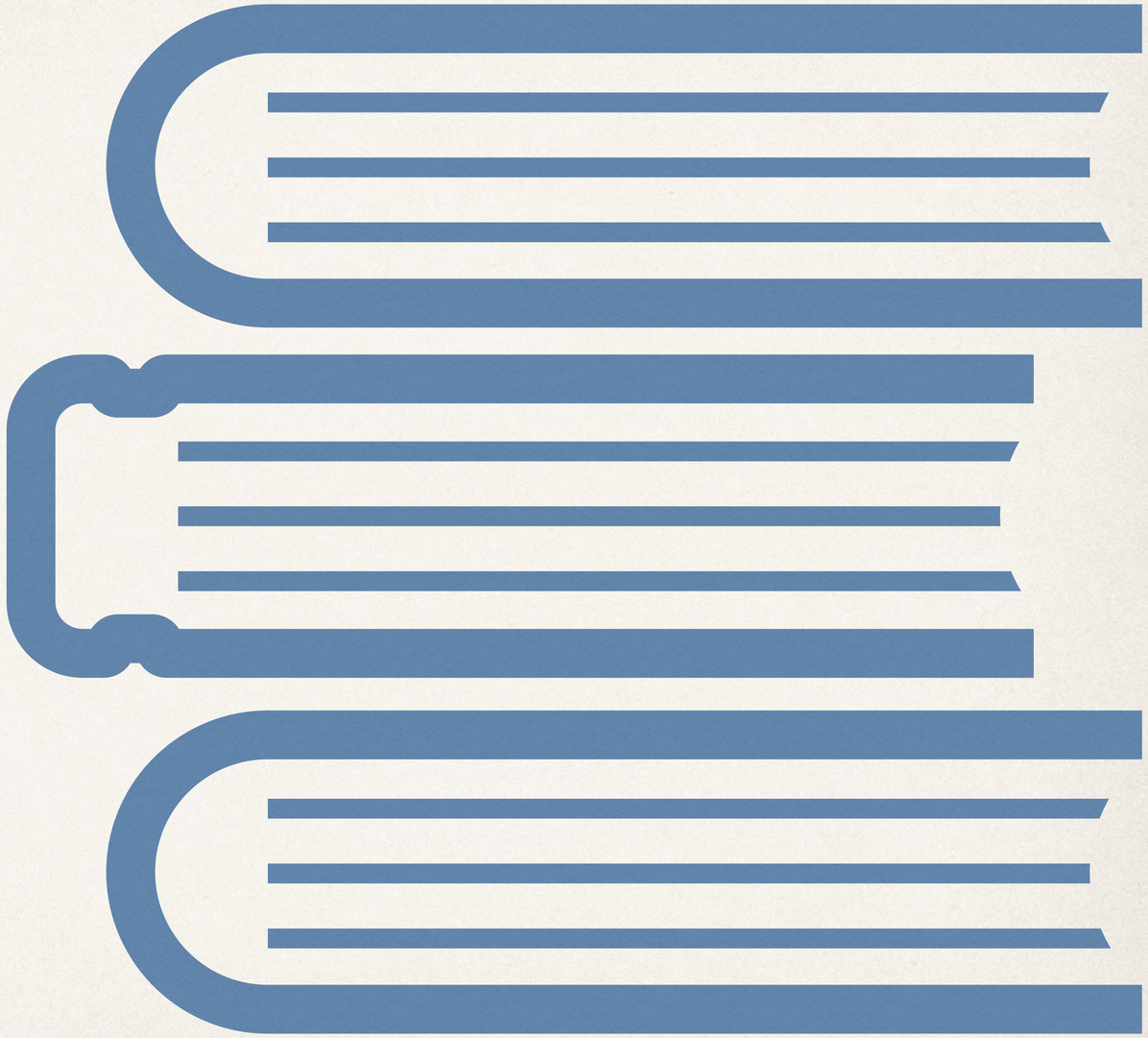
GitHub

- One reason git is so popular is the existence of GitHub a website that allows you to create a free account to use their servers as your repository.
- This is also a great place to find other peoples repositories, that is there code that you can download and use.
- Like for example the notebooks and slides used in this class.

https://github.com/ari-maller/bootcamp_notebooks

Exercise 3

Create a GitHub account if you don't have one. Create a repo and then clone it to your local machine.



git workflow

- git init - create an new repository / or create a repository on GitHub using the web interface.
- git clone <repo url> - make a local copy of the repository
- git add <filename> - adds file or updates file to local copy
- git commit - updates changes to the repository
- git push - uploads changes to remote repository
- git pull - to pull any changes from the repository back to your local machine

Git Basics

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via `HTTP` or `SSH`.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

Undoing Changes

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

Rewriting Git History

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, a branch name, a tag, or a relative reference to `HEAD`.
`git reflog`	Show a log of changes to the local repository's `HEAD`. Add `--relative-date` flag to show date info or `--all` to show all refs.

Git Branches

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

Remote Repositories

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

Atlassian

Visit atlassian.com/git for more information, training, and tutorials