**GitHub**

**Creating a new Jupyter Notebook File in a Repo Folder**

* Using the anaconda prompt
  + navigate to the cloned folder in the cmd prompt
  + Jupyter notebook
    - this will open Jupyter notebook and saving, will keep the file in the proper directory

**Cloning a Repository**

* You add a directory on your computer to sync with the GitHub repo
* Steps
  + Click “Clone or Download” then copy the url
  + In a command prompt:
    - Navigate to the folder to store the repo files
      * useful commands:
        + ‘pwd’ shows the working directory
        + ‘cd’ for change directory
        + ‘d:’ to switch drives
    - Clone the repo
      * git clone (then paste the copied url)
      * may need to turn off controlled folder access under ransomware protection

**Viewing Files in a Repo**

* git ls-files
  + views files in the working directory
* ls
  + can try, but looks like it doesn’t work in conda prompt

**Viewing Contents of a File**

* cat *path/to/file*
  + displays the contents of the file

**Repo Settings**

* git config --list
  + shows the settings
    - --system # for every user
    - --global # for every one of my projects
    - --local # for a specific project
* **Should always set your name and email address**
  + git config --global *setting.name setting.value* # replace the *setting.name* and *setting.value*
    - user.name
    - user.email

**Syncing Computer Folder to a Repository (and checking for changes)**

* The four primary commands for updating and checking the repo are in **bold**
* **git status**
  + typed in the conda cmd prompt will show any “untracked files” that have not been uploaded to GitHub
* git diff [*info*]
  + shows changes to your repository with no *info* passed
  + shows changes to a *directory* if that is passed
  + shows changes to a *filename* if that is passed
  + git diff *ID1..ID2*
    - use two different commits separated by two dots to see what changed between two commits
    - ID1 can be HEAD and ID2 HEAD~1
      * HEAD is the most recent
      * HEAD~1 is one before that
      * HEAD~2 is two before that and so on
* git annotate *file*
  + shows who made the last change to each line of a file and when
* **git add *filename***
  + will track a file and add it to a commit (that will need to be pushed to GitHub)
* git reset HEAD *path/to/file*
  + will unstage a file that has been git added
* git checkout -- *filename*
  + will delete (permanently) any changes to a file that have not yet been git added to the staging area or have been removed from the staging area
* git checkout *hash* *filename*
  + will restore a file to the *hash* (commit hash) you specify
    - usually only need the first several chars of the hash
    - does so as a new commit, so you can undo your undo by restoring a different commit
* **git commit -m “*message*”**
  + your message goes in double quotes (in anaconda prompt) and is a way to identify and track commits (it can have spaces)
  + may get an error if your global email and global name aren’t set, so set them if necessary
  + still need to push the commit to GitHub
  + omit the -m “message” to open a text editor to provide more detailed info
    - ctrl o to save
    - ctrl x to exit
* **git push origin master**
  + to sync and save the changes
* git log [*info*]
  + shows changes to a repository, directory, or filename
  + git log -3 *filename*
    - will show you only the last 3 commits for your *filename*

**Syncing Repository to your Computer**

* git pull (paste the copied url from clone/download button)
  + will pull files from GitHub to your computer if you have already cloned the repo

**Ignoring Files When Committing**

* Create a file called in the root dir of the repo .gitignore
* .gitignore
  + specifies files and dirs. to ignore
  + can use wildcards to specify a pattern to ignore
    - build # ignores any file or dir called build
    - \*.mpl # ignores any file with .mpl extension
    - pdf # ignores file or dir called pdf, but not .pdf extension