**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
  + Note: a new folder will be created in this directory that contains the files
  + You should have a source for the repo directory, but don’t create the actual repo directory
    - this is done when you clone in using the steps below
* See section below for turning an existing directory into a repo
  + that section, however, is NOT used to sync to GitHub
* Steps when using GitHub
  + Click “Clone or Download” on the GitHub page 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
* Steps when cloning from one repo to another just on your machine
  + Navigate to the folder to store the new copy/clone in the cmd prompt
  + git clone */existing/project* [*newprojectname*]
    - will call the root dir the same as the original, unless you specify a *newprojectname*
* Viewing the source of a cloned repo
  + git remote [-v]
    - when executed, will show a list of remotes (a repo can can from more than 1 source)
    - optional verbose (-v) flag will display URL[s] if a repo is from GitHub or a similar service
    - the “origin” points to the original directory
  + git remote add *remote-name URL*
    - adds a remote under the *remote-name* supplied to the specified local or web *URL*
  + git remote rm *remote-name*
    - removes the specified remote

**Creating a New Repo (using cmd prompt)**

* git init *project-name*
  + *project-name* is the name you want the new repo’s root directory to have
  + avoid creating a repo in another repo, this is allowed but becomes very complicated

**Convert Existing Folder to a Repo**

* git init
  + run in the project’s root directory (can also git init *path/to/dir*) if in any other directory

**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, editing a repo)**

* 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
    - ID1 and ID2 can be different branches
* 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 add . # adds every file to a commit
* 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 rm *filename*
  + will remove a file and stage the removal of that file (git add) all in one step
* **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
    - ctrl m for ms-dos format
      * look for questions that need answering, ctrl-*char*
* **git push origin master**
  + to sync and save the changes
  + actual syntax is **git puch *remote-name branch-name***
    - for basic GitHub/Springboard, the remote is origin and the branch is master
* 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**

* GitHub
  + git pull (paste the copied url from clone/download button)
    - git pull origin master # to pull from the original repo on GitHub
      * add flag --no-edit to keep the editor from opening
        + git pull --no-edit origin master
    - navigate to the proper cmd dir first
    - will pull files from GitHub to your computer if you have already cloned the repo
* CMD
  + git pull *remote branch*
    - navigate to the proper cmd dir first
    - will pull files/dirs from the *branch* (master or other) associated with the *remote* supplied into the current dir
* Conflicts
  + if you try to overwrite changes you have made on the local system when pulling, git stops you
  + you must commit local changes or revert them before you can try to pull again

**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

**Branches**

* git branch
  + provides a list of all the branches for a repository (\* next to the branch you’re in)
* git diff *branch-1..branch-2*
  + displays the differences between two branches
* git checkout *branch-name*
  + switches to a new branch
    - can only do this if all changes have been committed
* git branch
  + creates a new branch (may need to supply the branch name as an arg)
* git checkout -b *branch-name*
  + creates a new branch and switches to it immediately
  + branch is created identical to the current branch until changes are made
* git merge *source desination*
  + merges the two branches
  + changes made in the *source* and incorporated into the *destination*
    - can result in a new commit in the destination that includes everything from the source
* **Handling conflicts when merging branches**
  + git status
    - after merging will remind you which files have conflicts
    - will show files in which both branches have been modified
  + nano *filename*
    - will open the file for editing
    - make appropriate changes
    - ctrl-o to save
    - ctrl-m for msdos
    - ctrl-x to exit
  + git add *filename*
  + git commit -m “*message*”