CS-181/DA-210 Setup

Anaconda Install

If you already have Anaconda installed on your computer, use the following instructions to upgrade, otherwise, skip ahead to the **New Installation** instructions below.

Upgrade an Existing Install

If you already have Anaconda installed, you should upgrade to the most recent version of the packages that are compatible with Python 3.7. The easiest way to do this is to open a Terminal (MacOS and Linux) or *Anaconda** Powershell Prompt (Windows) command line window. If you have trouble with the command-line on your particular platform, you can launch Ananconda-Navigator, then launch JupyterLab, click the left-hand "plus" symbol (or, from the menu, File->New Launcher), and then click on "Terminal".

From the command-line, enter the following:

```
$ conda update conda
$ conda install python=3.7
$ conda update ananconda
$ conda update jupyterlab
```

Note that, in showing command line commands, the `$` represents the command line prompt, which is often prefixed with text showing the host, the current user, and/or the current working directory. You **do not** type the dollar sign when entering the command … only the text that follows.

Check the set of packages that will be installed and, if there is no risk to your current environment, go ahead and hit or type `Y` followed by to proceed with the install.

Once the install is complete, you should check the version on, at least, the `jupyterlab` package. As of the time of this writing, the version used by your instructor is `1.1.4` , and can be checked by the following:

```
$ conda list jupyterlab
```

For me, this results in the following output:

```
jupyterlab               2.1.5                    py_0
jupyterlab_server        1.2.0                    py_0
```

The version of a package is in the second column.

If you need to keep a current configuration and create a separate environment to use for your CS-181/DA-210 work, please see your instructor, possibly setting up a time during office hours to complete a separate installation.

New Installation of Anaconda

Navigate to https://www.anaconda.com/distribution and download the version appropriate to your platform.

When the download is complete, open the downloaded file to begin the installation.

- When in doubt, or if not an advanced user, select the defaults.
- A typical installation installs for *only* the current user in a location underneath the user's *home* directory.

Portions of this install can take significant time. On Windows, the step "Setting up the package cache" is quite lengthy. Do not cancel the operation. You can click on the "Details" button to see the ongoing progress.

If you need to uninstall, see the Anaconda web pages, as they have specific instructions for each platform.

Command/Terminal Window

After installation of Anaconda, you should be able to perform command-line operations. For Mac users, these occur in the `Terminal` application. For Windows users, you should use the `Anaconda Powershell Prompt` application installed as part of Anaconda. Based on your platform, use the following instructions to launch a command-line window and then execute the command: `conda list` to verify your installation.

Do **not** skip this step, as our cloning of the class repository will happen through this command-line interface.

Mac platform

- Click on the magnifying-glass icon and search for **Terminal** and accept the found result
- When you see a "command prompt" (line ending in `$` with a cursor insertion point following) type in `conda list`
- Tip: Right-Click on the `Terminal` icon in the dock and select "Options->Keep In Dock"

Window platform

- Navigate in the Start Menu to `Anaconda Powershell Prompt`
  - When you see a "command prompt" (line ending in `$` with a cursor insertion point following) type in `conda list`
- Tip: Add the Anaconda Bash to your dock/applications bar

If you get a list of packages with package version, then Anaconda was installed properly. If not, let your instructor know.

Launch Jupyter Lab

The installation will also have given you a GUI, known as `Anaconda Navigator`, by which you can launch the Jupyter Lab notebook development environment. This runs in a tab in your default browser.

When using the GUI, you can shut down the development environment and return all resources by going to the `File` menu in the Jupyter Lab browser tab and selecting `Shut Down` and then closing the browser window. You should **always** cleanly exit by performing this sequence.

You can also launch Jupyter Lab by the following command in a Terminal/Anaconda Powershell Prompt command-line:

```
$ jupyter lab
```

As with the GUI launch, you shut down the envvironment and return all resources by going to the `File` menu and selecting `Shut Down` and then closing the browser window. You should **always** cleanly exit by performing this sequence.

Make sure you can launch Jupyter Lab by either the Navigator or by the command line.

Packages

Given a package name, we can issue commands in the Command/Terminal window to add packages into our anaconda environment by issuing the commands:

```
$ conda install mysql-connector-python
$ conda install requests-oauthlib
```

Conda ecosystem packages can also be installed using the Anaconda Navigator application.

We will also need the following pip package, which is not part of the conda ecosystem,

```
$ pip install ipython-sql
```

Installation of `Git`

Git is the software that underlies GitHub as well as other systems of software that support controlled version control and distribution of collections of files. We will be using Git and GitHub as our primary means of providing files/documents/notebooks to be used for class work and for homework.

The software is a collection of command-line and system functions, and does not, by itself, provide a graphical interface. We install it first so that subsequent steps have the software we need.

Windows Platform

The `git` software for the windows platform can be obtained from

https://gitforwindows.org/

and contains both the underlying software, as well as a GUI for dealing with repositories and a version of a shell to give a more powerful command-line interface.

Download the software and run the resultant download. If you already use git repositories and/or have needs outside of this class for uploading new versions of files to GitHub, you may want to customize some of your responses during the install. Otherwise, the defaults should be sufficient for the needs of the CS-181/DA-210 class.

Once installed, you will see an entry in the Start Menu that contains shortcuts for launching the GUI, and for launching the shell, known as *Git Bash*.

Mac/Linux Platforms

You can check if git is already installed by opening a Terminal window and using the command-line:

```
$ git --version
```

If the result is something like the following:

```
git version 2.21.0
```

then `git` is already installed. If the version is recent, you need not modify your git installation. If you get a response like `Command Not Found`, or if the git version is way out of date, you should install git. You can also choose to install a GUI that helps use git, but a git GUI is beyond the scope of these instructions.

Navigate to https://git-scm.com/downloads and download the version of git, as appropriate for your OS X or Linux platform by clicking the name of your platform. If needed, click the "start download manually" and proceeding through the download screen. When the download is complete, open the downloaded file and follow the instructions.

If you need to manage multiple repositories and to upload/push/commit changes, you should go to a command-line and set your git username. This course will only be using a single student repository and you will not be updating the repository, so this step should not be necessary unless you have exceptional circumstances. The command, if needed, is as follows:

```
$ git config --global user.name "J Doe"
```

GitHub

`GitHub` is an organization that hosts project repositories and is used by professionals and students to collaborate and share files/software/documentation and using `Git` to allow coordination and version control for those activities.

If you do not already have one, set up a `GitHub` account/identity. You can get a free account that is restricted to open source and public repositories, or you can sign up for a "Student Pack" that provides access to some additional software and resources. While we will not be using these additional software packs,

- General free GitHub account signup: GitHub Login
- Students Pack signup: GitHub Student Pack

Unless you have a justifiable reason not to, use your Denison email address. The GitHub username can be anything you like, as long as it is not already taken, but please be appropriate, because your instructor will be using this username to *invite* you to gain access to one or more course student repositories.

You will have to establish that you are a human, not a robot, and to verify your email as part of the process of creating a GitHub identity.

There will be a Bulletin post on Notebowl where you will respond in a comment, giving your GitHub userid. Your professor will use this to invite you to the class repository, which will be how we distribute all homework assignments and documentation.

You will receive, by email, an invitation to the repository. Follow the link and accept the invitation **before** you proceed to the next step.

Local Folder Organization and Repository Clone

We encourage you to *think* and *organize* before you dive in and start creating class-based files and directories/folders in a haphazard way on your local computer.

As we should have learned from our Intro CS course, each user logged in to a computer has a **Home** directory[1]. If your login username is *cooper*, our home directory on a Mac system might be:

```
/Users/cooper
```

while on a Linux system it might be

```
/users/cooper
```

or

```
/home/cooper
```

and on a Windows machine, it might be

```
\\Users\cooper
```

The string we use to refer to a directory or a file is known as its **path**. Different systems use different separators between directory and file elements in the path (`\` versus `/`). Some systems also permit special characters or sequences to have meaning in interpreting a path. - `~` typically can be used as a substitute for the current logged in user's home directory - `..` means the parent of the "current" directory, where the notion of a current directory is maintained by the engine executing code (like a Python script or notebook) or the Terminal or Command window in which the path specification is being used.

One organization that makes sense for local files on your computer supporting multiple classes in a college setting is, within your home directory, use a directory for each individual class that you take, like `cs112` and `cs181` . You might also decide, if you use a sync'd cloud storage solution, to use a similar organization, but under `Dropbox` or something similar. You also, could choose to create your class directories under `Documents` or `My Documents` .

Please, **do not** combine files from multiple classes into the same directory. Also, once we have created a local copy (a clone) of the files being given to you as part of the class distribution, it would be best if you keep other class files *separate* from the class repository.

> On your local computer, decide on your organization and create directories based on what you have decided. For the following discussion, we will assume that your login is `cooper` , and you have decided on the above suggested organization, so the full path for your cs181 class materials would be

```
/Users/cooper/cs181
```

> for the above Mac OS example.
>
> How you accomplish this organizational creation depends on your platform. On Mac OS, you use a Finder window and can select the `Go` menu and then select `Home` to get a view on your home directory. On Windows, you open a Windows Explorer window (typically by clicking a Folder icon in the dock) and Navigate from `Computer` or `My PC` (the root of the file system) to your login/username folder. You can then use menu selection to create new folders and double click and repeat to create subfolders.

> Hint: To make the steps below easier, avoid spaces in the names of your directories, and avoid getting too complex an organization.

GitHub Clone of `denison-cs/cs181-f20` repository

One of the primary means of delivering material to students in this class will be through a GitHub repository named `cs181-f20` . When one creates a local copy of the directories and files in a Git repository for the *first time*, this action is called a **clone**. So our goal is to clone the repository and for the destination of the clone to be on your local computer within the cs181 folder you created above. There are two prerequsite steps:

1. Make sure you have registered and created your GitHub username.

2. Give your GitHub username to your instructor, who owns the cs181-f20 repository and needs to add you as a valid user of the repository.

> The current plan is for the instructor to be on GitHub on the machine at the front of the classroom, and to have students ready for this step come up to the instructor machine and add their GitHub username on the repository.

Now you are ready to use `Git` to clone the repository from GitHub.

1. Open a Terminal (on Mac) or Anaconda Powershell Prompt (on Windows) command line window.
    - Once open, type the command `pwd` (command for "print working directory") at the prompt. You should see the terminal respond with your home directory. From the perspective of the terminal, this is your "current" directory.
2. You need to get your terminal so that the current directory is your class folder, as created above. So relative to the result of `pwd`, the path to get to your class folder will depend on the organization you decided on above. For the example above, the relative path is `cs181` (note the absence of a leading slash, which would indicate a path that starts at the root of the filesystem).
    - Determine your relative path from the (home) directory obtained above, and your class folder. (i.e. what is your equivalent of `cs181`?)
    - In your terminal, type the command `cd` (command for chage directory) followed, on the some line, by the relative path to your class folder. So my command looks like:

```
$ cd cs181
```

    if the `cs181` folder/directory were created under `Documents`, the command might be:

```
$ cd Documents/cs181
```

- If I execute `pwd` in the terminal, I should now get the full path of my course directory. If you do, then you are ready to proceed to the next step.

3. Execute the following in your terminal:

```
$ git clone https://github.com/denison-cs/cs181-f20.git
```

    This step can only succeed if the prerequisite steps have all been successful.

    On successful execution, you will be asked for your GitHub username and password, and the system will report the Cloning and unpacking operations. The end result will be a directory named `cs181-f20` *within* your class folder. You should navigate into this directory and get a sense of its organization. Try doing the following in the terminal:

```
$ cd cs181-f20
$ ls
```

    (The `ls` is the command to *list* the contents of the current directory.) Does what you see make sense? You should see entries for `general`, a directory for files like these setup instructions, as well as a folder named `homework`, which is where all the files needed for you to complete homework assignments will be kept.

[1] The terms **directories** and **folders** refer to the same thing, a location in the file system on a computer in which we can organize both **files** as well as other (sub)directories in a hierarchical fashion. Those who work in the **systems** subfield of computer science use the term directory, while many users, based on their experience in the GUI metaphor of a desktop, use the term folder.