Getting started with Python and Git: Mac/Linux

Welcome to the University of Arizona Applied Math program! This document will provide instructions and tips on how to get set up with GitHub and Python when using a Mac or Linux machine, as well as how to get access to the files we will be using in the programming portion of the Integration Workshop.[1]

Contents

[1]Much of this document is adapted from the Getting Started with Python guides that are part of the Foundations of Applied Mathematics curriculum by Humphereys and Jarvis, available at `https://foundations-of-applied-mathematics.github.io`.

1 Git

Git is a *version control* tool that helps you keep track of changes to your files. It lets you save different versions of your work, switch between them, and easily revert to previous versions if needed. If you are using Linux, open your terminal emulator app (the default terminal app is different across distributions, but common terminal apps include `gnome-terminal`, `xterm`, `x-terminal-emulator`, etc.) and type the following command, followed by `Enter`. Typing a command and hitting enter will run the command. Note that the first `$` sign indicates a new line in your terminal, and is not part of the command.

```
$ sudo apt install git
```

If you are using MacOS, open your terminal emulator app (the default is the `Terminal` app) and type `git` and hit `Return`; if you don't have `git` installed, the terminal will prompt you to install it. If you have an M1 Mac or later, you should also double check that you have OS version 12 or later.

Github is a website where you can store your git projects online. It makes it easy to share your work and collaborate on projects, because it allows multiple people to work on the same files at the same time and keeps track of everyone's changes. It provides methods to easily integrate those changes together into the main version of the files. It's a widely-used tool in applied math and computational sciences, and many researchers publish their code bases on Github.

In Math 589, you will use Github to submit programming assignments and receive feedback, so it's important to familiarize yourself with it. You will *push* your code from your local machine to your online Github repository, and then the most current version of your code will be *pulled* by the instructor from your online repository to their local machine for grading. No one else will have access to your files until you push them to the remote repository.

1.1 Setting up GitHub

1. *Sign up for GitHub.* If you already have a GitHub account, sign in to it and continue to the next step. If not, create a GitHub account at `https://github.com/`.

2. *Set up your Git credentials.* You will need to set up your Git `user.name` and `user.email`. Your `user.name` should be your actual name, and `user.email` must be the email associated with your GitHub account. To do this, open your terminal and run the following:

   ```
   $ git config --global user.name "your name"
   $ git config --global user.email "your email"
   ```

3. *Create an SSH key.* This step only needs to be done once on each computer you want to use to acces your GitHub repositories. If you have multiple repositories on the same

computer, you do *not* need to repeat this for each one. To create an SSH key, enter the following command in your terminal:

```
$ ssh-keygen -t ecdsa -b 256
```

Press the `Enter` key to accept the default file location. It will then prompt you to enter a Password, but you can press `Enter` again to skip this step if you don't want a Password. The key will then be created. The file for the key will be placed in the `/home/<your username>/.ssh` directory.

4. *Add your SSH key to your GitHub account*:

   (a) From Github, click on your profile icon in the upper right corner.

   (b) Click on "Settings" toward the bottom of the menu.

   (c) Click on "SSH and GPG keys" on the left.

   (d) Click the green "New SSH key" button.

   (e) Make a title for this key (it doesn't matter what you put, but it may be a good idea to specify which machine this key belongs to)

   (f) Make sure **Key type** is set to `Authentication Key.`

   (g) Using the file explorer, nagivate to the `.ssh` directory.

   (h) Open the *public key* file. This file should be called `id_ecdsa.pub`; do *NOT* use `id_ecdsa` (without the `.pub` extension). Copy the contents of this file and paste it into the `Key` field on GitHub.

   (i) If you're having trouble navigating to the `.ssh` folder in explorer, you can print out the contents of the file by running this command in your terminal:

   ```
   $ cat ~/.ssh/id_ecdsa.pub
   ```

   (j) Once you've pasted the context of `id_ecdsa.pub` into GitHub, press the green **Add SSH key** button.

   (k) In your terminal, run

   ```
   $ ssh-add ~/.ssh_ecdsa
   ```

   If you get an error that says "Could not open a connection to your authentication agent," then run

   ```
   $ eval $(ssh-agent)
   $ ssh-add ~/.ssh/id_ecdsa
   ```

   (l) To verify that this worked, you should see your new ssh key in the file `~/.ssh/known_hosts`.

1.2 Cloning GitHub repositories

A GitHub repository is a Git project that is stored on GitHub. You can clone a repository to your own machine, make changes to the files locally and, depending on the permission

settings of the remote (online) repository, push changes back to the remote repository. We'll practice this by cloning the repository set up for this workshop.

1. In a browser, navigate to `https://github.com/jcallahan4/arizona_programming_workshop`
2. On the top right-hand side, click the green button that says "`<> Code`".
3. Copy the address in the box. Make sure it is an SSH address (it should begin with `git@github.com`). If not, click the "SSH" tab above the address bar.
4. In your terminal, navigate to the folder you want to store the repository files in, then run the command

   ```
   $ git clone <address you copied>
   ```

   This will clone the repository into a folder named `arizona_programming_workshop` (the name of the remote repository). Make sure to replace `<address you copied>` with the actual SSH address in the command.
5. You should now have a folder called `arizona_programming_workshop`, and inside should be some PDF files, a `projects` directory, and a file called `install_python.sh`.

1.3 Some Git commands and terminology

- *Adding files.* Git will only track files you specifically add. You can add files individually or they can be added all at once. To add a file, use the command `git add <filename>`.
- *Commits.* A commit is a snapshot of your code at a specific time. You use commits to record changes you've made to your code, and each commit has a unique identifier that you can use to reference your commits later. This makes it easy to revert your code to previous states. It's good practice to commit every time you leave your computer. To create a commit, run

  ```
  $ git commit -m "Commit message"
  ```

- *Branches.* A branch is a separate line of development that diverges from the main (sometimes called master) line of development. Creating a new branch allows you to work on a feature or bug without affecting the main codebase. You probably won't need to branch your code in Math 589 but it is an extremely useful feature, and it's good to be aware of it for the future.
- *Pushing and pulling.* These are likely the two most frequent operations you will use. When you *push* changes to a remote repository, you're sending your commits to the central server, hosted on GitHub, where others can access them. This is where the instructor will access your code. To push your changes, first create a commit, and then run

```
$ git push <remote> <branch>
```

Here, "remote" refers to the remote repository where you want to push your changes (e.g., "origin") and "branch" refers to the name of the branch you're pushing (e.g., "main"). See below for further discussion. It is good practice to push every few hours of working and every time you're done with your code.

When you *pull* changes from a remote repository, you're downloading changes that others have made and incorporating them into your local codebase. If the instructor adds feedback files or makes changes to your code, they will be pushed to the remote repository and you can pull them to your own computer from there. To pull, use the command

```
$ git pull <remote> <branch>
```

Good practice is to pull just before any time you push.

- *Origin and main.* "Origin" refers to the default remote repository where your code is stored. When you clone a repository, Git sets up a remote called "origin" that points to the URL of the original repository. You can push/pull changes to/from "origin" to collaborate with others.

  The name of the default branch in a Git repository is "main". This is the branch where the main line of development occurs, and it's typically the branch you'll be working on most of the time. However, depending on the repository, the default branch may be named something else (e.g., "master").

  Here is an example of pushing after making some changes.

```
$ git add -u
$ git commit -m "Changed requirements.txt file to include
   new packages"
$ git push origin main
```

  The first line tells git to add any tracked files that have been changed to the next commit. This is referred to as *staging*. The second line creates a commit, and the third line pushes the commit to the remote repository. To see which files have been staged, which are tracked, and other useful information, you can run the command `git status`.

2 Installing Python

Before installing Python on Mac, you will need to install Homebrew (`brew`). To check if `brew` is already installed, run

```
$ which brew
```

If the output is

```
/usr/local/bin/brew
```

then you can skip to the update/ugrade step. If the answer is

```
brew not found
```

then you need to first install `brew` by running the command found at `brew.sh`, which will probably look like

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
   Homebrew/install/HEAD/install.sh)"
```

Once `brew` is installed, update it by running

```
$ brew update
$ brew upgrade
```

To install Python, navigate in your terminal to the workshop directory you just cloned (`arizona_programming_workshop`). You can do this with

```
$ cd <path-to-directory>
```

Replace `path-to-directory` with the actual file path. For example, you might have cloned the repository to your desktop, in which case the file path would be

```
~/<your username>/Desktop/arizona_programming_workshop
```

Once there, run the command

```
$ bash install_python.sh
```

This command uses the program `bash` to execute the script `install_python.sh`, which will install Python for you.

3 Writing and running Python code

Writing Python code involves creating plaintext files with a `.py` extension, where you can define functions, classes, scripts, etc. These Python files contain the instructions that tell the computer what to do. These instructions are translated into machine code by the Python interpreter. To do this, we execute our Python file using the Python interpreter, which we access from the command line. The syntax is simple. In your terminal, navigate your to workshop directory. Then, run the command

```
$ python simple_script.py
```

This executes the code written in `simple_script.py`. Now, open `simple_script.py` in your favorite text editor. Play around with the values and numbers in the script, save it,

and run it again to see how the output of the script changes.

3.1 Text editors

There are many text editor programs which come packaged with special features that make writing code much easier than the text editors that come default with MacOS and Linux (`TextEdit`, `Gedit`, etc.). We recommend using VSCode, though it's not required. It's free and open source, and it integrates very well with Git and Python. You can download it at `https://code.visualstudio.com/`. Once you have it installed, you can search for it on your machine under `Visual Studio Code`, or you can open it from your terminal using the command `code`.

VSCode has lots of useful extensions, and you should explore the `Extensions` tab (the little building blocks icon on the left) to see what's on offer. One particularly useful extension is the `Latex Workshop` extension that can make it much easier to write LaTeX(believe me, you will want to make your life easier when it comes to LaTeX!). You can read the documentation on the extension website.

3.2 A simple Python script

In your favorite text editor, open up a new file, and name it `hello_world.py`. in this file, write the following code:

```
if __name__ == "__main__":
    print("hello world!")
```

Then save and exit the file. Then, in your terminal, run

```
$ python hello_world.py
```

You should see the message "hello world!" printed to the terminal. In Python, the line

```
if __name__ == "__main__":
```

is a way to check if the script is being run directly. When this check is true, it means the script was started by the user. When you write a Python script, it's best practice to put all the code you want executed under this check. Then next line uses the print command to tell the computer to print out whatever is inside the parentheses to the terminal. Congratulations, you're now a Python programmer!

3.3 The Python interactive environment

Python also comes with an interactive interface, also known as the Python shell or repl (readd-eval-print loop). The Python shell allows you to run code one line at a time. This makes it easy to check how a specific command works, test out small code snippets, or

experiment with an idea without needing to write a full program. To use the Python shell, in your terminal run the command

```
$ python
```

After an initial welcome message, you should see a prompt that looks like this:

```
>>>
```

On that line, execute the following command (note that the >>> indicates the prompt and isn't something you need to type)

```
>>> print("hello world!")
```

You should see the following output:

```
>>> print("hello world!")
hello world!
```

You can use the interactive shell to perform arithmetic, assign variables, or run any other Python commands of your choice. To exit the shell, run

```
>>> exit()
```

and you will be returned to your normal terminal interface.

4 Python packages

Python packages are collections of modules and code that extend the functionality of Python, allowing you to add new features and capabilities to your projects. They can include pre-built functions, classes, and tools for a wide range of tasks, from data analysis to web development. However, sometimes installing them can be tricky, so we manage Python packages using Python environments.

4.1 Python environments

A Python environment is a self-contained directory that contains a specific version of Python and a set of packages. When you install a Python package, it gets installed to the directory of your active environment. It helps you manage dependencies for different projects and avoid conflicts between them.

Python environments create isolated spaces for your projects. This means each project can have its own set of dependencies, without interference from other projects. Without environments, all projects share the same global Python interpreter and packages, leading to conflicts when different projects require different versions of the same package.

By using environments, you can precisely control the versions of Python and packages

used in a project. This makes it easier to reproduce results, whether you're sharing your code with collaborators or running it on a different machine.

Creating an Environment

Python comes prepackaged with a method for creating environments, called the `venv` module. Here are the steps to create a virtual environment using `venv`.

1. Use terminal to navigate to the folder you'd like to store your environment in. For example, you might use

```
$ cd ~/<your username>/Desktop/python_environments
```

2. Once there, use the following command to create an environment:

```
$ python -m venv math589
```

   Here, you can replace `math589` with whatever you'd like to call your environment. This command creates a directory named `math589` containing the virtual environment.

3. Activate your new environment with the following command:

```
$ source math589/bin/activate
```

   If you named your environment something different from `math589`, make the appropriate substitution. For this command to work, you'll need to be in the same directory that you placed your environment folder in (to see this, try navigating to a different folder and running this command). Once activated, your command line prompt will typically change to include the environment name, indicating that you are now using the environment. You will now have access to every Python package and module that you have installed in this environment.

4. To deactivate the environment, simply use the command:

```
$ deactivate
```

   This will revert your shell back to the global Python environment. Unlike the activation command, you can use this command from any directory in terminal.

4.2 Installing Python packages

To install a Python package into your Python environment, we use a package manager called `pip`. When you install a package using `pip`, it tracks what you have installed and what needs to be installed to make the package work, and then manages all these dependencies and requirements for you. The process is simple. As an example, we will install the Python package `numpy`, which is a very widely using scientific computing package and one which we will use heavily in the workshop.

1. In terminal, navigate to the directory where your environments are stored
2. Activate your environment

```
$ source math589/bin/activate
```

3. Use `pip` to install `numpy`:

```
$ pip install numpy
```

4. Check if `numpy` installed correctly:
   - Open your interactive Python shell by executing the command `python` in the terminal
   - Try to import numpy into Python using the following command:

     ```
     >>> import numpy
     ```

     If the command executes without any error, you have successfully installed `numpy`!

Python comes bundled with many useful packages, like `math` or `time`. There are also a great many Python packages available through `pip`. Another package we will use in the workshop will be the `scipy` package. `scipy` is built on `numpy` and offers additional tools for optimization, integration, linear algebra, statistics, and more. Follow the instructions above to install `scipy` by replacing `numpy` with `scipy` in the installation command.

4.3 Importing and using Python packages

To use a Python package in your code, you first need to import it. You do this by including a line (most commonly at the top of your code file) that looks like this:

```
import <package-name>
```

Then, when you want to use a function that belongs to that package, you include it in your code like this:

```
<package-name>.<function-name>
```

For example, if I wanted to find the absolute value of a number, I could do the following in the interactive shell:

```
>>> import math
>>> absolute_val = math.abs(9 - 10)
>>> print(absolute_val)
1
```

This is a silly example, but it demonstrates how to call functions from packages.

Another useful feature of imports is the ability to create an *alias* for a package or module using the `as` keyword. This is typically used to import a package but use a shorter name for it in your code, which is more convenient. For example, it is common to give `numpy` the alias `np`:

```
>>> import numpy as np
>>> new_array = np.array([1., 2., 3., 4., 5.])
>>> print(new_array)
[1., 2., 3.]
```

Submodules

Many packages provide their functions and classes in *submodules*, which are like sub-packages that are dedicated to specific tasks. For example, `scipy` has many submodules like `linalg`, which is dedicated to linear algebra, and `stats`, which is dedicated to statistics. You can import these submodules just like you would a regular package like so:

```
>>> import scipy.linalg as la
>>> import numpy as np
>>> x = np.array([4., 0.])
>>> y = np.array([0., 3.])
>>> la.norm(x - y)
5.0
```

Additional Tips

1. Don't install packages into your base environment (i.e., without activating a specific environment first). This will keep installed packages from modifying your system files and prevent conflicts.

2. It's a good practice to use a descriptive name for your environments, reflecting the project or the purpose of the environment.

3. Always activate the appropriate environment before installing new packages or running your code, to ensure that you're working in the correct context.

5 Additional Resources

If you have the time and inclination, here are some additional resources you can look through for more information on the subjects covered here.

5.1 Git

- Official Git tutorial
- Github Git cheat sheet
- Codeacademy Git lesson

5.2 Python

- Included in the workshop repo is a PDF file titled *Python Essentials*, which contains several labs designed to help you get comfortable with Python. We will be going through much of the same material in the workshop that is covered in the lab workbook, but the lab workbook contains much more detailed information with more ways to practice. If you have time before the workshop and feel uncomfortable with your Python skills, we *highly* recommend you work through the first three labs in the workbook.
- Learn Python the Hard Way
- Python Cheat Sheet

5.3 Unix shell (Terminal)

- Unix shell cheat sheet
- Codeacademy Command Line lesson – 4 hours
- Including in the workshop repo is a PDF file title *Data Science Essentials*. It contains several labs designed to familiarize you with data science tools. There are two Unix shell labs, which will help you get familiar with the terminal and using the command line. They don't take too long and are very useful.