# 12-770 Fall 2024: Lecture #2: Setting up your Development Environment

Mario Bergés

2025-01-16

Here is **what** I intend to cover today:

- Basics of Python
- What is Interactive Python (IPython)?
- What are Jupyter Notebooks?
- What are version control systems, and what is Git?
- What is GitHub?
- How do I share code through a version control system like Git?

At the end of this process, I would like for each of you to be able to create an Jupyter Notebook locally on your computer, and then be able to allow anyone else in the course (if pushing it to our course's repository) or in the world (if pushing it to a public repository) to see it.

This very same file we have on the screen now will make that journey.

## Before you begin

Things you'll need to do ahead of time:

1. Create an account on github.com
2. Install the Anaconda Python distribution
3. Install git on your computer, which you can get here

In addition to the references posted on the slide deck, here are some references that will be **very** helpful to ensure you understand what we are doing and how it all works:

1. Git References
    - What it is and what is it used for?
        - Official Documentation, especially the first three videos on this page.

  - Official Git Tutorial, if you are already familiar with the command line interface to some other version control software and just need to get started.
- How does it work?
  - Visual, interactive explanation: this is really valuable if you want to wrap your head around the basic of what's happening under the hood.
  - Git from the inside out: an in-depth discussion of how things really work under the hood.

2. Python References

- Why Python and how is it useful for for Scientific Computing?
  - First, a quick intro to Python for Scientific Computing
  - Other cool sources:
    * SciPy lecture notes: heavy focus on Python fundamentals.
    * Quantitative Economics with Python: the name says it all.
    * Online Courses:
      · Introduction to Scientifi Python (Stanford)
      · Practical Data Science (CMU)
      · Computational Statistics in Python (Duke)
- How to use Python?
  - Here's a tutorial.
  - Two libraries that we are going to be making extensive use of are numpy and matplotlib. The same person who wrote the tutorial above has tutorials for numpy and matplotlib

OK. Let's get you started.

---

## Python Basics

Python is a popular programming language known for its simplicity and versatility. It is widely used in engineering, data analysis, and research. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

### Why use Python?

- Readable and easy to learn.
- Extensive libraries and frameworks (e.g., NumPy, pandas, scikit-learn).
- Strong community support.

---

## Interactive Python (IPython) and Jupyter Notebooks

**IPython** provides an enhanced interactive Python shell. It is a foundational component of **Jupyter Notebooks**, which are interactive documents combining code, text, and visualizations.

**Why use Jupyter Notebooks?**

- Ideal for teaching, data exploration, and prototyping.
- Supports inline visualizations.
- Allows sharing of code and results in a readable format.

---

## Version Control and Git

### What is version control?

Version control systems (VCS) track changes to files over time. They allow collaboration and the ability to revert to previous versions.

### What is Git?

Git is a distributed version control system that lets multiple developers work on the same project without conflict. Key features include:

- Branching and merging.
- Distributed workflows.
- High performance and security.

### What is GitHub?

GitHub is a cloud-based platform for hosting Git repositories. It provides tools for collaboration, issue tracking, and project management.

---

### Cloning the course's git repository

This file you are currently viewing is part of the course's git repository, which you can find here:

https://git.inferlab.org/websites/12-770

You could either clone it using the command line interface to git, or a graphical user interface (whichever you installed on your computer if you chose to install git). From the command line, for instance, you would issue this command to clone it:

```
git clone https://git.inferlab.org/websites/12-770.git
```

Make sure that you can clone the repository in your computer by issuing that command.

If you are successful, you will be able to see a new folder called `12-770` inside the folder where you issued the command. A copy of this Jupyter Notebook file should be in there as well, under lectures/code and you can view it by opening an Jupyter Notebook Server as follows:

```
jupyter notebook
```

Just make sure you issue this last command on the corresponding folder.

## Creating and using your own repositories

The steps we followed above were for cloning the course's official repository. However, you will want to repeat these steps for any other repository you may be interested in working with, especially the ones that you end up creating under your Github account. Thus, let's practice importing one of your repositories.

Follow these steps:

1. Head over to github.com and log in using your credentials.
2. Create a new repository and name it whatever you like.
3. At the end of the process you will be given a checkout string. Copy that.
4. Use the checkout string to replace the one we used earlier that looked like this: `git clone http://github.com/yourusername/yourrepository.git`
5. Try issuing that command on your computer (obviously, replacing `yourusername` and `yourrepository` with the right information)
6. If all goes well, you'll have your (empty) repository available for use in your computer.

Now it's time for you to practice some of your recently learned git skills.

Create a new Jupyter notebook, making sure to place it inside the folder of the repository you just cloned.

Add a couple of Python commands to it, or some comments, and save it.

Now go back to the terminal and add, commit and push the changes to your repository:

```
git add yourfile.ipynb
git commit -m "Made my first commit"
git push origin main
```

If this worked, you should be able to see the file added to your repository by simply pointing your browser to:

`http://github.com/yourusername/yourrepository`

## Doing away with the terminal

Because Jupyter can be used to issue commands to a shell, directly, you can avoid having to switch to a terminal screen if you want to. This means we could have performed all of the above git manipulation directly from this notebook. The trick is to create a *Code* cell (the default type of cells) in the Jupyter notebook and then issuing the commands preceded by a `!` sign, as follows:

```
!git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    ../../assignments/S24-12-770-Assignment-1-solutions.html
    deleted:    ../../assignments/S24-12-770-Assignment-1-solutions.ipynb
    deleted:    ../../assignments/S24-12-770-Assignment-1-solutions.pdf
    deleted:    ../../assignments/S24-12-770-Assignment-1-solutions.qmd
    modified:   Lecture_2_Setting_Up_Your_Environment.qmd

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   Lecture_2_Setting_Up_Your_Environment.html
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Lecture_2_Setting_Up_Your_Environment.quarto_ipynb
    Lecture_2_Setting_Up_Your_Environment_files/
```

Try running the above cell and see what you get.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
x = list(range(0,10))
```

```
x
```

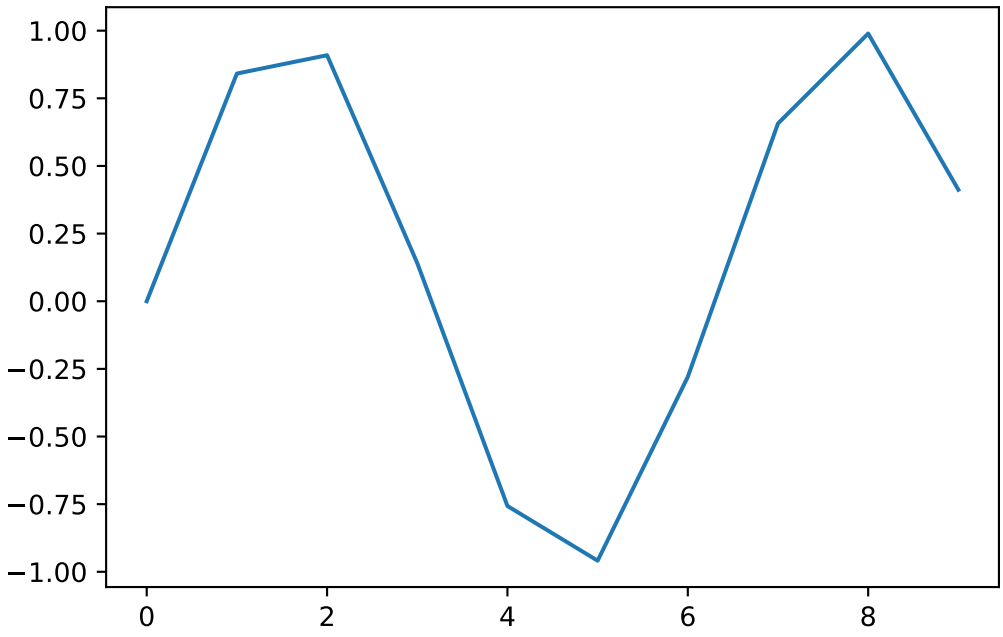
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
y=np.sin(x)
```

```
y
```

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

```
plt.plot(x,y)
```




---

## Introducing `uv`: A Modern Python Environment Manager

While Anaconda is a powerful tool for managing Python environments, `uv` offers a lightweight and modern alternative. It integrates well with existing tools and is ideal for project-based workflows.

### Why use `uv`?

- Simple and fast.
- Seamless environment creation and management.
- Works with any Python installation.

### Getting Started with `uv`

1. Install `uv`:

```
pip install uv
```

2. Create a new environment:
   ```
   uv new my-env
   ```
3. Activate the environment:
   ```
   uv activate my-env
   ```
4. Add dependencies:
   ```
   uv add numpy pandas
   ```
5. Deactivate the environment:
   ```
   uv deactivate
   ```

### Transitioning from Anaconda to `uv`

If you are familiar with Anaconda, you will find `uv` straightforward. For example:

- Creating an environment: `uv new` replaces `conda create`.
- Activating an environment: `uv activate` replaces `conda activate`.
- Managing dependencies: `uv add` replaces `conda install`.

---

## Wrapping Up

By the end of this session, you should be able to:

- Understand the basics of Python and Jupyter Notebooks.
- Use Git and GitHub for version control.
- Use either Anaconda or `uv` for managing Python environments.

For further reading and practice:

- Python Documentation
- Jupyter Project
- Git Documentation
- GitHub Guides
- uv Documentation

Happy coding!