**Jupyter slideshow:** This notebook can be displayed as slides. To view it as a slideshow in your browser, type the following into the console:

> ipython nbconvert [this\_notebook.ipynb] --to slides --post serve

To toggle off the slideshow cell formatting, click the CellToolbar button, then View --> Cell Toolbar --> None.



Your Development Environment[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Your-Development-Environment)

*Authors: Kiefer Katovich (San Francisco), Dave Yerrington (San Francisco), Sam Stack (Washington, D.C.)*

Student Pre-Work[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Student-Pre-Work)

**Mac**

* Install [Homebrew](http://brew.sh/).
* Install Git (after installing Homebrew, type "brew install git").

**Windows**

* Install [Git Bash](https://git-for-windows.github.io/)

Learning Objectives[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Learning-Objectives)

Part 1: Using the Command Line[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Part-1:-Using-the-Command-Line)

*After this lesson, you will be able to:*

* Create folders and files using the command line (mkdir, touch).
* Change directories and list directory content (cd, ls).
* Check the current working directory (pwd).

Part 2: Git and GitHub[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Part-2:-Git-and-GitHub)

*After this lesson, you will be able to:*

* Use and explain common Git commands, including init, add, commit, push, pull, and clone.
* Distinguish between local and remote repositories.
* Create, copy, and delete repositories locally or on GitHub.
* Clone remote repositories.
* Establish Secure Shell connections to remote repositories.

Lesson Guide[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Lesson-Guide)

[Part 1: Using the Command Line](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#command)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Part-1:-Using-the-Command-Line)

* [Introduction](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#introduction)
* [Command Line](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#command_line)
* [Paths](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#paths)
* [Editing and Examining Files](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#editing_files)
* [Finding Files](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#finding_files)
* [Independent Practice](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#independent_practice)
* [Environments for Data Science](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#ide)
* [Conclusion](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#conclusion)

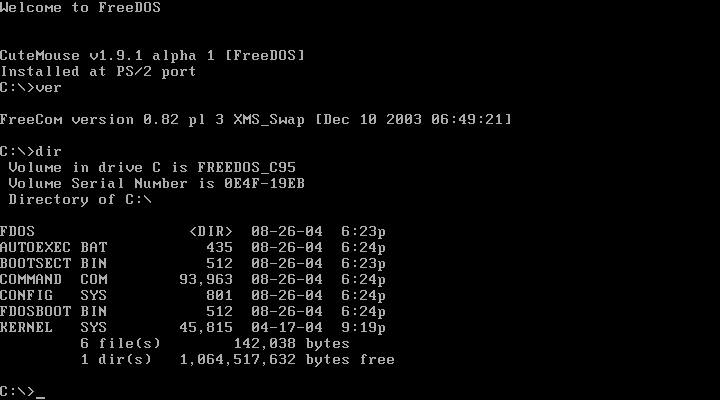
[Part 2: Git and GitHub](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#git_github)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Part-2:-Git-and-GitHub)

* [Introduction](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#introduction2)
* [Git Basics](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#git_basics)
* [GitHub & GitHub Enterprise Accounts](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#gh_ghe_accounts)
* [Creating and Cloning Repos](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#making_cloning)
* [Pulling](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#pulling)
* [Secure Shell (SSH)](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#ssh)
* [Independent Practice](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#independent_practice2)
* [Conclusion](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#conclusion2)

Using the Command Line[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Using-the-Command-Line)

What is a GUI?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-is-a-GUI?)

There was a time when computers didn't come with a graphical user interface (GUI, pronounced "gooey"). Instead, everyone interacted with the computer using text commands in what we call a command-line interface (CLI).

*Image from Wikimedia*

Today, the command line still exists, even though you may have never seen it as a casual computer user. However, knowing basic CLI commands is a useful skill.

This CLI is the gateway to your operating system and can perform many of the same actions without a graphical interface. It is also facilitated by what we call a "shell."

Why Command Line?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Why-Command-Line?)

Everything you can do in a windowed environment, you can do in the terminal — and faster. When you use the mouse, click on buttons, confirm operations, or wait for windows to load, you waste time! Terminal commands can assist you with:

* Running processes
* Finding files
* Substring match of file contents
* Assessing performance
* Remote operations
* Web browsing
* Installing packages
* Managing your development environment

What Is a Shell?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-Is-a-Shell?)

A shell is a type of command-line program that contains a simple, text-based user interface, enabling us to access all of an operating system's services. It is, put simply, a program that accepts text as an input and translates that text into the appropriate functions you want your computer to run.

*Taken from Just for Fun:* [*Type like a hacker*](http://hackertyper.com/)

Running applications, checking settings such as free hard drive space, and even web browsing can all happen from the shell.

**Windows Users**

In Windows, you can access a shell in several ways:

* **Windows Command Prompt:** A legacy DOS-based shell. Press Win-R to open a Run dialog. Then, enter cmd to open the shell.
* **Windows PowerShell:** The official Windows-native shell and scripting language, intended to replace the antiquated Command Prompt.
* **Windows Subsystem for Linux:** New to Windows 10, this allows you to run Ubuntu (a popular Linux distribution) natively in Windows. [You must turn on this Windows feature manually.](https://docs.microsoft.com/en-us/windows/wsl/install-win10)

**Windows Users**

In this class, we will use a popular UNIX shell called bash. If you have not already, we recommend installing [Git Bash](https://git-for-windows.github.io/).

**What Is Git Bash?** Git Bash provides a set of executables that emulate bash commands in the Windows shells.

**Why?** Since UNIX is the prefered operating system for programmers and developers alike, almost all of the lessons at GA have been written for interaction with UNIX.

**Pro Tip:** UNIX commands and Windows commands are not polar opposites — in fact, most of their commands and interactions are quite similar.

Forget Finder — Get Fast at Using Your Laptop[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Forget-Finder-—-Get-Fast-at-Using-Your-Laptop)

Opening and Closing Terminal[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Opening-and-Closing-Terminal)

First, we need to launch the command prompt. We do this by using Spotlight:

* ⌘ (Command) + Space
* "Terminal"
* Enter

Notice that you can actually hit enter as soon as the field autocompletes. Get used to taking shortcuts — don't type the whole word out if you don't have to, and avoid using your mouse if you can open or use an app with just keyboard shortcuts. It may seem hard right now, but, when you get used to it, it will save you hours of time cumulatively.

Paths[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Paths)

Every file or folder in a file system can be read, written, and deleted by referencing its position inside that system. When we talk about the position of a file or a folder in a file system, we refer to its "path." There are two different kinds of paths we can use to refer to a file — **absolute paths** and **relative paths**.

**Directory** is an important term that is typically used interchangeably with *folder*. Technically, a folder contains the files themselves, whereas a directory is just a listing of them (e.g. a student directory). That said, when we say "navigate to your project directory," think of it as "navigate to your project folder."

What Is an Absolute Path?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-Is-an-Absolute-Path?)

All files can created, updated, or deleted using the command line interface. We do this by referencing **paths**; either relative or absolute.

An absolute path is the specific location of a file or folder as accessed from the root directory, typically shown as /. The root directory is the starting point from which all other folders are defined and is not usually the same as your **home** directory, which is normally found at /Users/[Your Username].

Working With UNIX Commands and File Paths[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Working-With-UNIX-Commands-and-File-Paths)

Typing **cd** — a command for "change directory" — with no parameters takes us to our home directory.

cd

If we type in pwd — a command for "print working directory" — from that folder, we can see where we are in relation to the root directory. The pwd command will always give you the absolute path of your current location.

Here's an example of absolute path:

open /Users/Lucy/desktop/a/b/c/file.txt

Notice that this path starts from / directory, which is a root directory for every Linux/Unix machine.

What Is a Relative Path?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-Is-a-Relative-Path?)

A relative path is a reference to a file or folder **relative** to your current position or the present working directory (pwd). If we are in the folder /a/b/ and we want to open the file that has the absolute path /a/b/c/file.txt, we can simply type:

open c/file.txt

or

open ./c/file.txt

We can also use the absolute path at any time by adding a slash to the beginning of the relative path. The absolute path is the same for a file or a folder, regardless of the current working directory, but relative paths differ based on directory. Directory structures are laid out like so: directory/subdirectory/subsubdirectory.

**Check:** What is the difference between an absolute path and a relative path?

Navigating Using the Command Prompt[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Navigating-Using-the-Command-Prompt)

* Changing directories
* Listing files
* Creating directories and files
* Removing files

The tilde (~) character is an alias for your home directory. Use it to quickly return home.

cd ~

Or, even more simply, you can type:

cd

The tilde is useful for shortening paths that would otherwise be absolute. For example, to navigate to your desktop, you can type:

cd ~/Desktop

Special Navigation commands[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Special-Navigation-commands)

The dot (.) means "right here".

The double dot (..) means "one directory up". For example:

The ls command lists files and directories in the current folder.

ls

It can also be used to list files located in any directory. For example, to list your applications, you can type:

ls ~/Desktop

To make a new directory, type:

mkdir folder

To create a new file, type:

touch file1

To remove a file, type:

rm file1

General Format for Commands[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#General-Format-for-Commands)

<command> -<options> <arguments>

* <command> is the action we want the computer to take.
* <options> (or "flags") modify the behavior of the command.
* <arguments> are the things we want the command to act on.

Using Wildcards in the Command Prompt[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Using-Wildcards-in-the-Command-Prompt)

The wildcard symbol (\*) is useful for using commands to operate on multiple files. To provide an example, first create a folder on your desktop and add some files.

mkdir ~/Desktop/example\_folder

cd ~/Desktop/example\_folder

touch cat.txt

touch dog.txt

touch bird.txt

touch fish.txt

You can then use the wildcard \* to operate on subsets of files. List any file with "i" in the file name, for example:

ls \*i\*

Or, remove any file with "d":

rm \*d\*

ls

Now - Try to delete your example\_folder

OK - so directories are different

rm -r example\_folder

If you have files in the folder you will be prompted before deleting each one.

rm -rf example\_folder

In the above example, the "example\_folder" directory, along with all files and directories within that directory, would be deleted with no prompt or message.

How Do You Know How a Command is Used?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#How-Do-You-Know-How-a-Command-is-Used?)

To learn how to use a command (or to refresh your memory), all commands have help pages.

On Mac/Linux

man <command>

On Windows/Git Bash (works for most commands

<command> --help

Pick a command and see what you can figure out. We'll use ls as an example in a moment

Hidden Directories Can Also Be Found[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Hidden-Directories-Can-Also-Be-Found)

There are hidden directories all over your file system — mainly to save you from youself. Using the parameters -lha to ls, we can find these directories.

ls -lha

-l:    One entry per line.

-h:    When used with the -l option, use unit suffixes: byte, kilobyte, megabyte, etc.

-a:    Include directory entries whose names begin with a dot (.).

Editing and Examining Files[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Editing-and-Examining-Files)

At times it's helpful to edit files in a pinch. We can accomplish this by using the terminal editor nano.

Use the following syntax to edit files from the terminal with nano:

nano [filename]

These hotkeys are available:

* **ctrl-w**: Search within file.
* **ctrl-o**: Save file as [filename].
* **ctrl-x**: Exit editor.

*The bottom of the editor contains the most common operations.*

Editing a file[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Editing-a-file)

Using nano, write the following to fish.txt

nano fish.txt

This is the first line!

This is second.

The third line comes next.

And now four.

Echo File Content to the Terminal[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Echo-File-Content-to-the-Terminal)

Sometimes it's nice to view the contents of files as text. There are a variety of ways to do this. The commands cat, head, and tail will allow us to view the entire or partial contents of a target file.

The cat command is typically used to quickly display short files. (cat is also used for concatenating files, hence its name!)

cat /etc/passwd

Traditionally, the /etc/passwd file is used to keep track of every registered user with access to a system.

**Only the first few lines of a file**

This command is useful when looking at files that might be too large to open in a traditional editor such as Sublime or Atom.

head /etc/passwd

**Only the last few lines of a file**

tail /etc/passwd

You can also pass the paramter -n to head and tail to control the amount of output displayed.

Searching Inside Files: grep[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Searching-Inside-Files:-grep)

The grep command will search within files and traverse within subdirectories.

**Find all files with the word "the" inside.**

grep -r "the" \*

Omitting -r will cause grep to only look within the current subdirectory. Using -i will make grep ignore the casing of characters, but at the expense of efficiency.

Finding Files[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Finding-Files)

By far, the most useful operation from the terminal is finding files. locate finds files all over your file system. The find command will find files relative to the current working directory but needs to be used in conjunction with a pipe operation.

Finding All Notebook Files Within Subdirectories of the Current Working Directory[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Finding-All-Notebook-Files-Within-Subdirectories-of-the-Current-Working-Directory)

find . | grep ipynb

Finding Specific File(s) Within the Entire System[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Finding-Specific-File(s)-Within-the-Entire-System)

locate nanorc

Finding Specific File(s) With a Substring Match[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Finding-Specific-File(s)-With-a-Substring-Match)

locate log

Miscellaneous Trick: Counting the Number of Lines in a File[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Miscellaneous-Trick:-Counting-the-Number-of-Lines-in-a-File)

At times, you may not want to load an entire file in memory. Using a combination of head, tail, cat, and a new command, wc, you can quickly assess the various size characteristics of any file.

Find the number of lines in a file:[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Find-the-number-of-lines-in-a-file:)

cat /etc/passwd | wc -l

Find the number of words in a file:[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Find-the-number-of-words-in-a-file:)

cat /etc/test.txt | wc -w

Have Questions About "piping"?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Have-Questions-About-"piping"?)

Here's some optional (but highly recommended) reading about pipe and I/O redirection on the command line:

* [I/O redirection](http://linuxcommand.org/lc3_lts0070.php)
* [Good examples of piping commands together](http://unix.stackexchange.com/questions/30759/whats-a-good-example-of-piping-commands-together)

Independent Practice[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Independent-Practice)

Try out the mkdir, touch, cd, pwd, and ls commands on your own. If you want, try out using the wildcard command as well.

Intro to Development Environments[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Intro-to-Development-Environments)

In addition to being able to write commands in the Unix OS(mac) or Windows OS(PC) command langauge in the terminal, we can also execute commands in a variaty of languages like Python, Java and Git in terminal/command line as well.

In your terminal, you can enter into a Python shell by simply typing python.

Within the Python shell, we can execute Python expressions

>>> *# assigning a variable*

>>> x = 'hello world'

>>> *# printing a variables contents*

>>> **print**(x)

hello world

Writing and trouble shooting alot of code in the terminal can be tedious, as it is hard to write several line scripts. Almost all developers don't actually write their scripts in the command line; instead they use text editors or development environments to write their code.

Try writing a for loop in the Python shell

listo = [1, 5, 9]

**for** item **in** listo:

**print**(itm)

We made an error in the second line of the for loop but we still have to rewrite the entire loop and we can't go back and just edit out mistake inline.

Common Environments for Data Science[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Common-Environments-for-Data-Science)

The Anaconda distribution we installed earlier comes with two useful Python-based development environments, Spyder and Jupyter. A common third-party environment is PyCharm.

**Jupyter Notebooks**

Jupyter uses cell based execution, which means you can run all the code in a cell simultaneously. Jupyter notebooks also have markdown and slide show integration, which means they make great blog and instructional resources! *All of the lessons in this class have been written in a Jupyter Notebook*.

Jupyter Notebooks open in your default browser and can be opened from the Anaconda Navigator or from the command line by executing jupyter notebook.

* [28 Jupyter Notebook tips, tricks and shortcuts](https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/)
* [Markdown Cheatsheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet)

**Spyder IDE (Integrated Development Environment)**

Spyder has a selection-based execution which allows you to run all the code that you have *selected* simultaneously. Spyder is very similar to R Studio, supports Jupyter notebooks, and has several customizable windows for displaying output, variables, and computer usage.

Syder is desktop software that opens in its own window. It can be opened from the Anaconda Navigator or from the command line by executing spyder.

* [Introduction to Spyder (Video)](https://www.youtube.com/watch?v=8JiWEZEnJ40)

**PyCharm IDE (by JetBrains)**

PyCharm is an excellent fully-featured commercial IDE for writing Python code files. It has a free-of-charge Community Edition which has no restrictions. PyCharm is often used for developing larger Python applications. However, it offers powerful features that other environments lack such as debugging capabilities, intelligent code refactoring, and integration with Git.

* [Free PyCharm Community Edition](https://www.jetbrains.com/pycharm/download)

An IDE (Integrated Development Environment) is a program that provides an all-in-one environment to programmers. For example, often in development you will open many programs all at once, e.g. Finder, a text editor, debugging terminal, a terminal window for displaying output, and a graphics editor. An IDE will provide all of these (and more!) inside a single application. That said, IDEs are often sluggish. So, many professional Python programmers prefer a plain text editor.

None of these environments is better than the other; you will even find many Python developers who only use a text editor to edit their code! That said, often Jupyter Notebook is preferred for quick experimentation and initial data exploration, whereas Spyder and Pycharm are preferred for writing non-notebook code files.

Text Editors[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Text-Editors)

In addition to IDEs, developers also use text editors to create or edit code and files. Text editors or more commonly used for files that are executed via the command line, as well as for software and website development.

Some common text editors that you may see or use include

* [Sublime](https://www.sublimetext.com/)
* [Atom](https://atom.io/)
* [Notepad++](https://notepad-plus-plus.org/) (Windows)
* [Vim](http://www.vim.org/)

Independent Practice[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Independent-Practice)

*Instructor Note: Demo opening a Jupyter Notebook and Spyder Notebook*

Open up a Jupyter Notebook with the Anaconda Navigator or command line and familiarze yourself with environment.

* Execute some Python code
* Add and remove cells
* Change cell type
* Look at the Cell, Kernel and Insert tabs

Open up a Spyder Notebook with the Anaconda Navigator or command line and familiarize yourself with environment.

* Execute some Python code
* Look at the Editor, iPython Console and Variable Explorer pane

*if you dont see them go to view -> panes*

**Python Version**

Please use this time to also check which version of Python you are running.

in your terminal type

python -V

Check the upper right hand corner of your Jupyter Notebook as well.

This class uses Python 3.

Even if you installed the Python 3 version of Anaconda, you can still easily run code in Python 2.7 by [creating a new Anaconda environment](https://conda.io/docs/user-guide/tasks/manage-python.html#installing-a-different-version-of-python).

For example, to create a new Python 2 environment called py27, from the command line run

*# Adding Python 2 Kernel*

conda create -n py27 python=2.7 ipykernel

source activate py27

conda install notebook ipykernel

ipython kernel install --user

Once the environment is created, you can select it within Jupyter notebook and use Python 2 as your kernel.

Lesson Review : Command Line[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Lesson-Review-:-Command-Line)

Today, we learned about the command line and some of its common commands. We also learned about file structures, absolute paths, and relative paths.

Additionally, we reviewed running Python in the command line as well as in other environments. There is a [Jupyter Notebook exercise](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/IpythonNotebookPractice/ipynb_practice1.ipynb) available if you want to practice working in the Jupyter IDE.

In time, you might find that simple operations are actually faster to perform from the command line.

Lesson Break[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Lesson-Break)

Git vs. GitHub and Version Control, an Introduction[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Git-vs.-GitHub-and-Version-Control,-an-Introduction)

First things first — Git is not GitHub. This is a common mistake people make.

What is Git?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-is-Git?)

[Git](https://git-scm.com/) is:

* A program you run from the command line.
* A distributed version control system.

Programmers use Git so that they can keep a history of all changes made to their code. This means that they can roll back changes (or switch to older versions) as far back as when they started using Git in their project.

A code base in Git is referred to as a **repository**, or **repo**, for short.

Git was created by Linus Torvalds, the creator of the Linux kernel.

What is GitHub?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-is-GitHub?)

[GitHub](https://github.com/) is:

* A hosting service for Git repositories.
* A web interface to explore Git repositories.
* A social network of programmers.

Some other points to note:

* We all have individual GitHub accounts and will be storing our code there.
* You can follow users and star your favorite projects.
* Developers frequently use Github to share and collaborate on open-source code.
* GitHub uses Git.

What is GitHub Enterprise (GHE)?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-is-GitHub-Enterprise-(GHE)?)

[GitHub Enterprise](https://enterprise.github.com/home) Is:

* Professional application of GitHub
* All repository data is stored on private and/or local machines and networks.

Where GitHub is the *public*, 'Social Network' for programming and programmers, Github Enterprise is the *private*, professional application of GitHub. Because, GitHub and GitHub Enterprise have a similar structure and are based off the git language, interacting with the two is is almost identical.

**Additional information on** [**GitHub.com vs GitHub Enterprise**](https://enterprise.github.com/downloads/en/comvsenterprise-082415.pdf)

Over the course of this class we will be interacting with both GitHub and GitHub Enterprise(GHE). You will set up a GHE account for sharing course materials and as a private repo for you to store your own works-in-progress.

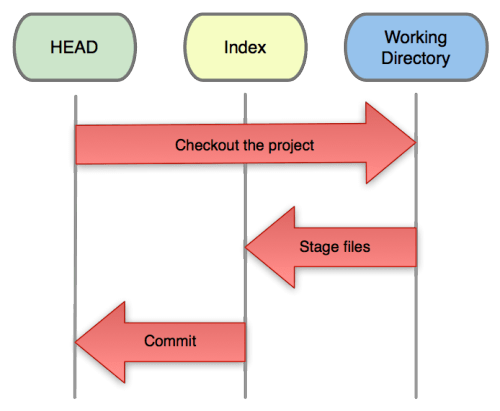
You will also setup a GitHub account as a location for you to host projects and work that you want to exhibit or share.

Can you use Git without GitHub?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Can-you-use-Git-without-GitHub?)

Think about this quote: “Git is software. GitHub is a company that happens to use Git software.” So yes, you can certainly use Git without GitHub!

Your local repository consists of three "trees" that are maintained by Git.

* **Working Directory**: holds the actual files.
* **Index**: acts as a staging area.
* **HEAD**: points to the last commit you've made.



What about commands?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What-about-commands?)

There are also a variety of commands you can use in Git. You can take a look at a list of the available commands by running:

$ git help -a

Even though there are lots of commands, in the course, we will really only need about 10.

Let's Git into it!: Code-Along (20 min)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Let's-Git-into-it!:-Code-Along-(20-min))

First, create a directory on your Desktop.

$ cd ~/Desktop

$ mkdir hello-world

You can place this directory under Git revision control using the following command:

$ cd hello-world *# don't forget to CD into the folder.*

$ git init

Git will reply:

Initialized empty Git repository in <location>

You've now initialized the working directory.

The .git folder[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#The-.git-folder)

We can look at the contents of this empty folder using this command:

ls -A

We should see that there is now a hidden folder called .git. This is where all of the information about your repository is stored. There is no need for you to make any changes to this folder. You can control all of the Git flow using git commands.

Add a file[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Add-a-file)

Let's create a new file.

$ touch a.txt

If we run git status, we should get:

On branch master

Initial commit

Untracked files:

(use "git add <file>..." to include in what will be committed)

a.txt

nothing added to commit but untracked files present (use "git add" to track)

This means that there is a new, **untracked** file. Next, tell Git to take a snapshot of the contents of all files under the current directory. (Note the .).

$ git add a.txt

This snapshot is now stored in a temporary staging area, which Git calls the "index." To confirm the files are staged and ready to be committed, again run git status.

You can alternatively add *all* new and modified files at once using the command below. This is not recommended, because you can accidentally add extra files if you are not careful! However, sometimes it is useful if you are adding many files at once and carefully use git status for verification.

$ git add .

Commit[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Commit)

To permanently store the contents of the index in the repository, (i.e., commit these changes to the "HEAD"), you need to run the following command:

$ git commit -m "Please remember this file at this time"

You should now get:

[master (root-commit) b4faebd] Please remember this file at this time

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 a.txt

Checking the log[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Checking-the-log)

If we want to view the commit history, we can run:

git log

As a result, you should see:

\* b4faebd (HEAD, master) Please remember this file at this time

'Git' on GitHub and GitHub Enterprise:[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#'Git'-on-GitHub-and-GitHub-Enterprise:)

For those of you that have not already set up your GitHub and GitHub Enterprise accounts, lets take a minute to do that now.

Account creation for both is simple. Create a *Username*, and provide an *email* & *password*.

*Keep in mind while you* ***can*** *use the same email, username and password for both accounts, they are* ***wholly separate****.*

[**GitHub**](https://github.com/) *This is yours*

[**GitHub Enterprise for General Assembly**](https://git.generalassemb.ly/join?source=header) *This is ours*

You can use a GitHub account to create a GitHub Enterprise account, but that will be **your** enterprise; in that, you will not be able to use it to access the General Assembly Enterprise GitHub.

If in the future you join another GitHub Enterprise you will also need to create another account for said enterprise.

Making and Cloning Repositories: Code-Along (20 min)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Making-and-Cloning-Repositories:-Code-Along-(20-min))

When using Git, GitHub and GHE it is common to have your repositories in several locations. Typically when we use GitHub and GHE we will have two repository locations, **Remote** and **Local**.

* **Remote:** Repositories that are not stored in our current location/machine. Usually where we store the repo.
* **Local:** Repositories that are stored on our current machine. Usually where we work on the repo.

**Let's do this together:**

1. Go to your GitHub account.
2. On the right-hand side, hit the + button for New repository.
3. Name your repository hello-world.
   * **DO NOT initialize the repository with a README, .gitignore, or license.**
4. Click the big, green Create Repository button.

We now need to connect our local Git repository with our newly created remote repository on GitHub. We have to add a "remote" repository, an address where we can send our local files to be stored.

On the right-hand side of your GitHub there should be a green 'Clone or download' button. This button should reveal a tiny window with a URL. Copy the provided URL, which is the path to this remote repo.

*Make sure you changed directories into hello-world prior to running this*

git remote add origin https://github.com/<GITHUB-NAME>/hello-world.git

*At this point you* ***may*** *be prompted for a password, especially when using GHE*

Pushing to GitHub[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Pushing-to-GitHub)

In order to send files from our local machine to our remote repository on GitHub, we need to use the command git push. However, you also need to add the name of the remote repo — in this case, we called it origin — and the name of the branch, in this case master.

git push -u origin master

Refresh your GitHub web page, and your files should appear.

Note: The -u option tells Git where to push to. You only need to do this once. Afterwards, you can just type git push.

Creating a README.md file[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Creating-a-README.md-file)

Let's create a README.md file and push it to GitHub!

Any file ending with .md is a Markdown file -- a text file with optional [Markdown formatting](https://daringfireball.net/projects/markdown/syntax). On GitHub, the contents of the displayed directory's README.md is automatically displayed.

Create a new README.md text file and add some text. (We'll try the command-line text editor nano this time!)

nano README.md

Now, try using the same procedure as before to:

1. Add the new file to the staging area.
2. Verify the file is in the staging area, ready to be committed.
3. Commit the file.
4. Push the commits to GitHub.

Refresh your GitHub web page, and the new README.md file should appear. Take a look underneath the directory tree, and its contents will be automatically displayed.

Pulling from GitHub[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Pulling-from-GitHub)

Sometimes, other people will make commits and push them to GitHub. Locally, we will need to fetch these changes and merge them with our local files. To do both of these steps at once, run the pull command:

git pull origin master

This pulls from our remote origin to our local master branch.

Since our remote repo contains the same commits as we do locally, it will tell us everything is up-to-date.

By now, you have have noticed that you are being prompted quite a bit for your password when interacting with GitHub or GitHub Enterprise via Git. There are two ways to work around this.

[**Caching your password in Git**](https://help.github.com/articles/caching-your-github-password-in-git/)

or

**Using SSH and SSH Agent** (*recommended*)  
You can use the guide in the [SSH Setup Guide](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/SSH-setup.md) or those available on github

* [Working with SSH key passphrases](https://help.github.com/articles/working-with-ssh-key-passphrases/)
* [Generating a new SSH key and adding it to the ssh-agent](https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/)
* [Adding a new SSH key to your GitHub account](https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/)

Secure Shell (SSH)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Secure-Shell-(SSH))

SSH, or Secure SHell, is a common means of adding an additional layer of security. Simply put, SSH is used to establish authenticity between a client and a server so that a secure connection can be formed. This can be useful for secure file sharing or remote application access.

How SSH works[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#How-SSH-works)

The SSH process at a high level is relatively simple.

1. Client makes a request to the server.
2. Server responds asking for authentication.
3. Client provides authentication.
4. If authentication is correct, a connection is established.

**Note:** An SSH Agent can be used to avoid being prompted for a password every time you push or pull to Github.

Independent Practice (15 mins)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Independent-Practice-(15-mins))

Repeat the steps above, starting from the creation of the remote repo, to create a *local* and *remote* repository with your GitHub Enterprise account.

If you have extra time feel free to try to set up an SSH. If you have any issues there are [SSH troubleshooting guides](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#ssh_trouble) at the end of the lesson.

Cloning your first repository[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Cloning-your-first-repository)

Alternatively, you can clone this repo from your GitHub link to have it automatically configured in a new directory.

git clone https://github.com/github-username/hello-world.git

Now that everyone has a repository on GitHub, let's clone one!

Cloning allows you to make a local copy of a remote repository.

Navigate back to your Desktop, and **delete your hello-world repository**.

cd ~/Desktop

rm -rf hello-world

Now, ask the person sitting next to you for his or her GitHub name and navigate to that repository on GitHub.

https://www.github.com/<github-username>/hello-world

On the right-hand side, you will see the green button 'Clone or download'.

Clone that repo![¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Clone-that-repo!)

To retrieve the contents of the repo, all you need to do is:

$ git clone git@github.com/<partners-github-username>/hello-world.git

Git should reply:

Cloning into 'hello-world'...

remote: Counting objects: 3, **done**.

remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0

Receiving objects: 100% (3/3), **done**.

Checking connectivity... **done**.

You've now cloned your first repository! The process will be the same with GitHub Enterprise.

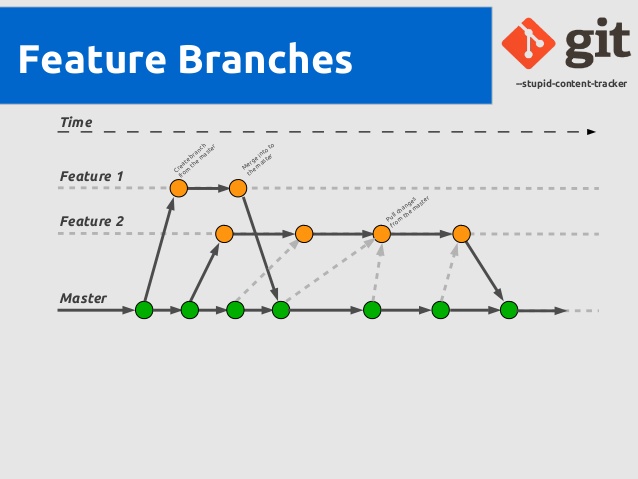
Cloning is must useful when there is a repo that exists remotely that we want on our local machines.

Create a Pull Request on GitHub: Code-Along (15 min)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Create-a-Pull-Request-on-GitHub:-Code-Along-(15-min))

What's a "Pull Request"?[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#What's-a-"Pull-Request"?)

Often times when we are working on a project with other collaborators we all can't work on the same repo at the same time so we have these things called *branches*. With branches we can create a branch off of the main repo or "Master Branch", perform our work whether that be additions or alterations and then merge the changes we made on *our* branch back into the master.

You can probably think of a few ways that multiple people trying to merge their work can get messy. Fortunately, we have Pull Requests as means of queing and validating merges. Rather than having your branch merge automatically, your request will go through an administrator who can review your changes and additions and approve or deny your request.



*Even though we are trying to* ***push*** *our information into the master, it is called a pull request because we are making a request to the administrator to* ***pull*** *or branch into the master.*

Creating a Pull Request[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Creating-a-Pull-Request)

Before you can open a pull request, you must create a branch in your local repository, commit to it, and push that branch to a repository or fork on GitHub.

1. Visit the repository you've pushed to.
2. Click "Branch:master" on the left-hand side and create a branch called 'branch-edits' by typing in the box and hitting enter. This should put you on your new branches page.
3. Make sure you're on your new branch and in GitHub create a new '.md' or '.txt' file using the 'Create new file' button. Call it what you like and place some text in it.
4. Click the "Compare & pull request" button in the repository pr.
5. You'll land right onto the compare page. Next, you can click "Edit" at the top to pick a new branch to merge to using the "Head Branch" dropdown menu.
6. Select the target branch that your branch should be merged to using the "Base Branch" dropdown menu.
7. Review your proposed change.
8. Hit "Click to create a pull request" for this comparison.
9. Enter a title and description for your pull request.
10. Click "Send pull request."

As you are working on your own repo in your own GitHub you will be able to merge the branch at your leisure.

GitHub Example Project[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#GitHub-Example-Project)

A company is building onto their website and adding a few new features. They are assigning a team of engineers to complete the task.

Engineer 1 is responsible for redoing the home page. Engineer 2 is responsible for reworking content pages 1 & 2 Engineer 3 is responsible for reworking content pages 3 & 4

In this situation the master branch would probably be a copy of the files and scripts for the original website. As this is the master copy it is best not to make any edits to it until those changes have been vetted.

The engineer team starts by making branch off of the master called 'project\_1'.

From 'project\_1' the team of engineers would probably create their own branches to perform their tasks in.

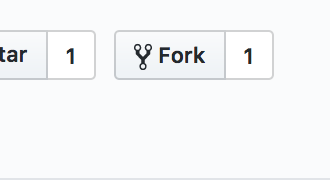
* Engineer 1 : 'project\_1\_homepage'
* Engineer 2 : 'project\_1\_content1\_2'
* Engineer 3 : 'project\_1\_content3\_4'

Once each engineer completed their task they would merge the changes from their branch into 'project\_1'.

Now that all of the individual work has been compiled they can test the website and make sure that there are no bugs or issues then merge 'project\_1' back into the master copy.

How we will use GitHub in this class.[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#How-we-will-use-GitHub-in-this-class.)

For the most part ,our use of GitHub in this class will be much more linear.

We will have a central class repository where all of the lesson materials are housed. You will create a Forked version of the master repo which will be yours to do as you please.  


* Forked repos are essentually personal copies of repositories.
* If the original is updated you can pull those changes to your copy via git pull upstream master.

Once we have a forked copy, we will clone the fork to our machines to make a local copy.

Most of your git commands will be pulling updates from the main class repo to your local and pushing changes make on your local to your remote version.

Assess: Independent Practice (10 min)[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Assess:-Independent-Practice-(10-min))

* With a partner take turns explaining the following commands. Pretend that they are unfamiliar with Git and this is their first exposure to it.
* init, add, commit, push, pull, and clone.

Lesson Review : Git and GitHub[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Lesson-Review-:-Git-and-GitHub)

Do you feel comfortable with Git, GitHub and GitHub Enterprise? As we'll be using them in alot a lot of our coursework, let's make sure you are!

We understand that Git and GitHub can be difficult to mentally grasp initially, so if you have an questions about the following or other Git aspects please ask now.

* Basic Git commands, init, add, commit, push, pull, and clone
* Local and Remote epositories
* Cloning, Branching, Merging and Pull Requests
* Installing Git

More useful Git tips[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#More-useful-Git-tips)

*Get the status relative to the remote version*

git remote update

git status

*Setup the upstream remote*

git remote add upstream <origin URL>

*Configure your name and email address*

git config --global user.name "John Doe"

git config --global user.email johndoe@example.com

Additional SSH Troubleshooting:[¶](http://localhost:8888/nbconvert/html/Desktop/SCHOOL_WORK/Week%202/your-development-environment-master/Your-Development-Environment.ipynb?download=false#Additional-SSH-Troubleshooting:)

**Access Denied**  
If you receive a message about "access denied," please notify an instructor or you can read [these instructions for diagnosing the issue.](https://help.github.com/articles/error-permission-denied-publickey/)

**HTTPS to SSH switch**  
If you're switching from HTTPS to SSH, you'll now need to update your remote repository URLs. For more information, see [Changing a remote's URL.](https://help.github.com/articles/changing-a-remote-s-url/)