**Command Line & Git Tutorials**

**I. Command Line Tutorial**

The command line is a text interface for your computer. It's a program that takes in commands, which it passes on to the computer's operating system to run.

From the command line, you can navigate through files and folders on your computer, just as you would with Finder on Mac OS or Windows Explorer on Windows. The difference is that the command line is fully text-based.

The advantage of using the command line is its power. You can run programs, write scripts to automate common tasks, and combine simple commands to handle difficult tasks - making it an important programming tool.

The tutorial below is for is for unix-based systems such as Linux and Mac OS X. For Windows users, you will need to Google equivalent commands.

**Opening your terminal**

On a Mac, go to Applications – Utilities – Terminal.

Once it’s open, you may want to pin it to your home screen since it’s so useful.

**Important Commands**

**1. pwd, as in “print working directory”**

Type pwd and then enter into your terminal. This should list your current working directory. Mine says “/Users/shareshianl” and yours probably says something similar. If you ever want to know which folder (aka directory) that you are currently in, you can type pwd.

**2. ls (lowercase L, lowercase S, as in “list”)**

Type ls and then enter. This lists all of the files in your current directory.

If you type “ls –a” this will list all of the files in your directory, including hidden ones. (If you scroll through, you’ll probably see files that start with a period. These are hidden files that you wouldn’t see from typing “ls” alone.

**3. cd (change directory)**

“cd” changes directories. You probably have a directory called Documents in your current working directory. Let’s move into it. To do so, type “cd Documents”

**4. mkdir (make directory)**

“mkdir” will make a new directory (aka folder) with the name that you specify. Now that we are inside Documents, let’s make a new folder called “terminal\_practice” (Note: You don’t want folder names to contain spaces so we’ll use the underscore instead.”) Type “mkdir terminal\_practice”

Type “ls” again and now you should see a new folder called terminal\_practice located there.

Just because you’ve make the directory doesn’t mean that you are inside it. Type “cd terminal\_practice” to now move inside that folder.

Type “ls” again and you’ll see that this is currently an empty directory.

**5. cd ..**

Let’s say you wanted to navigated back up a directory into Documents. Type “cd ..” and then enter.

Now type “pwd” and you’ll see that you are back in Documents. But actually, we want to be back in terminal\_practice so type cd terminal\_practice once again to get there.

**6. touch**

“touch” makes new files. Let’s make a new file called “helloworld.py” by using touch. Type “touch helloworld.py”

Type “ls” and you should now see the helloworld.py file there.

**6. cp (copy)**

Let’s say we wanted to copy this helloworld.py file into our Documents folder. I’ll type “cp helloworld.py ~/Documents” (Note: The format will always be “cp filename newlocation”)

**7. rm (remove)**

Let’s navigate up to Documents again by typing “cd ..”. We don’t actually want the helloworld.py file there so let’s remove it. To do so, type “rm helloworld.py”. If you type “ls” again you’ll notice the file is gone.

Note that if you wanted to remove a directory, not just a single file, you’d need to type “rm –r” before the directory name, not just “rm”. “-r” specifies the recursive option, which deletes everything within the directory.

8. **Up arrow**

Press the up arrow several times. Note that it lists the past commands that you’ve used. You can always navigate up to repeat a command instead of typing it all in.

**9. Tab key**

Let’s say we want to navigate back to the helloworld.py file located inside terminal\_practice. Instead of typing the whole file name, just type “cd term” and then press the tab key. You should now see the entire command listed, and you can now press enter.

**II. VIM and other text editors**

Type “ls” to see the helloworld.py file. This file is currently empty – let’s add stuff to it! To add stuff to it, we’ll need to use a text editor. There are many options – VIM, emacs, vi, Atom, Sublime, etc. VIM is one of the ones that comes pre-installed on a Mac so let’s use that.

**1. Opening vim**

Type “vim helloworld.py”

**2. Insert mode**

We need to be in the insert mode in order to edit the file. Type “i” to do so.

**3. Edit the file**

We can now edit the file. Let’s type the following:

print(‘Hello, World!’)

for i in range(10):

print(i)

**4. Save the file**

To save the changes, press escape and then type “ :w”

**5. Exit vim**

To exit vim, type “:q”

**6. Running Python in your terminal**

Okay, we don’t need Jupyter Notebooks or Anaconda or PyCharm or anything to run a Python program – we can do it directory from our terminal!

Type “python helloworld.py” and press enter

If you wrote your program correctly, you should see its output on the screen!

**Opening Jupyter Notebooks (or any program!) through your terminal**

We can now open Jupyter Notebooks through your terminal. Type “jupter notebook” and press enter.

To quit it, press Control-Shift-Z.

**III. SOME EXTENSIONS (THIS PART IS OPTIONAL)**

**1. Sublime Text**

Some of the best coders are so comfortable using the command line that they write ALL of their programs in VIM through the terminal. Crazy, right? If you aren’t that hardcore then I really like Sublime Text. You can open download it here:

<https://www.sublimetext.com/>

You can open it like you would open up any other application through Finder – Applications. You can then create a file as if you were typing into something like Word or Notepad, save it, and then run it from the command line. Just make sure that wherever you saved the Sublime text file is in the same place as your current working directory of your terminal when you go to run your code.

**2. Oh My Zsh**

All of the pros use Oh My Zsh, which makes using your shell (aka terminal) a lot easier to use and prettier. Go to <http://ohmyz.sh/> and follow the instructions for downloading (you can type the curl command directly into your terminal).

**IV. GITHUB INTRODUCTION**

It’s an exciting day, folks! We’re going to get you set up on GitHub. That way, you can put some of the programming projects you’ve done up on your GitHub website for the world to see. (One of the most important parts of a programming job or internship application is having some of your coding work visible on GitHub).   
In addition to making your code visible to prospective employers, here’s a scenario for why you’d want to use GitHub:

Let’s say you have a project you want to work on.   
Now you have **two primary motives:**  
**(a)**To write some code  
 **(b)**To take up the project as a team, meaning that more than one people are going to be responsible for writing the code.

Suppose you write a "Hello World!" program and then your friend/other developer decides that it should say "Hello, I am awesome" instead of the original "Hello World!".

Even for this little piece of code, you will have a copy on your computer and your friend will have one on his computer. Let’s say you change the code; now at this point your friend wouldn't know the changes you have made and vice versa.

This is where the Git magic comes.   
**1.**Git lets you save your code online.  
**2.**Git will allow all the developers of a project to see what changes the other one

**The “Git” in GitHub**

To understand GitHub, you must first have an understanding of Git. Git is an open-source version control system that was started by Linus Trovalds—the same person who created Linux. Git is similar to other version control systems— Subversion, CVS, and Mercurial to name a few.

So, Git is a version control system, but what does that mean? When developers create something (an app, for example), they make constant changes to the code, releasing new versions up to and after the first official (non-beta) release.

Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

**The “Hub” in GitHub**

We’ve established that Git is a version control system, similar but better than the many alternatives available. So, what makes GitHub so special? Git is a command-line tool, but the center around which all things involving Git revolve is the hub—GitHub.com—where developers store their projects and network with like minded people.

Let’s go over a few of the main reasons that geeks like to use GitHub, and learn some terminology along the way.

**V. GETTING STARTED WITH GITHUB**

**1. Install Homebrew**

Homebrew is a package manager. Once you have Homebrew installed, you can install other programs from the terminal, like Git.

A. To install Homebrew, go to <https://brew.sh/>

B. There should be a command prompt like the one below that the website tells you to copy and paste directly into your terminal. Do so and press enter.

“/usr/bin/ruby -e "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install)>"

C. It will tell you to press RETURN to continue.

D. Then, you’ll need to enter your system’s password (the password that you use to log into your computer when you turn it on)

E. You should receive an “installation successful message” in your terminal when the download is complete.

F. Update Homebrew by typing “brew update” into your terminal and pressing enter.

**2. Install Git**

Now we can use Homebrew to install git. Just type “brew install git” into your terminal.

**3**. **Setup a GitHub email account.**

Go to <https://github.com/> and sign up for GitHub. You’ll want to use an email address that you’ll have even after you graduate from OES, because you’ll probably be using this for a while, maybe a lifetime! You only need the free version and you don’t need to check any of the extra boxes.

**4. Start a GitHub project on the GitHub website**

Click on “Start a project” or the “+ sign – New Repository” in the top right hand corner of the page. Either one will bring you to the same page.

Let’s create a Repository name called “MTA\_Subway\_Project” and write in the description something like “Uses pandas to analyze MTA subway data.” Keep the project public and don’t click on the README box. Then click on “Create repository”.

The webpage that you’ll be brought to will give you instructions for creating a new repo. Keep this webpage open but don’t follow the instructions for now. Instead, do this next…

**5. Create a reposition of the same name on your local machine**

Okay, now on your personal computer, create a folder called “MTA\_Subway\_Project” in your Documents folder (note that this should be the EXACT SAME name as the one you set up in the last set on GitHub).

Copy ALL of the Python files you used for your MTA subway project, along with your presentation slides, into this folder.

**6. Create a README file**

It is best practice to always include a README.md file in your repo. This is a file that explains what your code is and how it works. Note that .md stands for “markdown”. Markdown is a way to style text on the web. If you want to read more about it (but you don’t have to) you can go to:

https://guides.github.com/features/mastering-markdown/

While you’re inside the MTA\_Subway\_Project folder, we can use vim from the command line to create this file. Type:

1. touch README.md *(then press enter)*

2. vim README.md *(then press enter)*

3. Type “i” to get into the insert mode

4. Write your file description. Something like “This program uses pandas to analyze MTA subway data to determine optimal placement for subway advertising.”

5. Hit escape and then type “:w” enter and then “:q” enter to save and exit vim.

**7. Synchronizing your local MTA folder with the GitHub web-based MTA folder**

Okay, these next four steps are the most important. TYPE THEM IN EXACTLY. If you make even the slightest error it can be very difficult unless you are a Git expert (which I’m not, so even I won’t be able to help you!) PROOFREAD EACH LINE 5 TIMES BEFORE YOU PRESS ENTER! Type:

1. git init *(then press enter)*

2. git add –A *(then press enter* - note this says add ALL the files in this folder toGitHub)

3. git commit –m “first commit” *(then press enter -* note that you can put whatever you want inside the quotation marks, but it is standard on the first commit to just say “first commit”

At this point, it might say something like “Your name and email address were configured automatically.” If it doesn’t, follow the terminal commands that they suggest in order to verify yourself.

4. git remote add origin <https://github.com/oesstudent/MTA_Subway_Project.git>

*(then press enter -* note that you can copy and paste the address from the GitHub website so you don’t make any errors)

5. git push – u origin master *(then press enter)*

At this point, it might ask for the GitHub username and password that you already set up.

**8. Look at your fancy pants webpage**

Return to the GitHub webpage that you have open and click on MTA\_Subway\_Project at the top of the page. You should now see your MTA files on this webpage. Everyone can see your wonderful work!

**9. Making Updates**

Okay, let’s say that you wanted to make updates to your MTA code or readme file. You can do that and resubmit the changes. Use VIM to go back into your README.md file and change one or two words around and resave and quit. Let’s now update Git with these changes. To do so, type:

1. git init

2. git add –A

3. git commit –m “updated readme”

4. git push –u origin master

**10. Adding other projects to Git**

You’ve done some other cool projects this year, too, like the teacher comment project and the banking project. I’d encourage you to put all of your projects on Git. For each one, you’ll need to follow the same steps again of setting up a new repo on the GitHub website, creating a folder of the same name on your local computer with your project files in it, and then synching them up.