VS Code Setup

**1. Download VS Code**



Download and install the latest version for your system from [https://code.visualstudio.com/ (Links to an external site.)](https://code.visualstudio.com/)

**2. Download the Extensions: Material Icon and**

**Prettier**

* Open VS Code
* Select the “Extensions” icon on the left (several squares stacked on top of each other).
* Type these needed extensions into search field and install them
  + Material Icon
  + Prettier

1. **Customizing VS Code**
   * To change the color theme and further customize VS Code, navigate to Preferences ->

Color Themes.

Command Line Basics

**Command Line Text Editors**

There are some built-in text-editors via the command line. Two common ones are **nano** and **Vim**. Some developers enjoy using them because they are free, always accessible on every computer, and use a very small amount of a computer system’s resources.

Don’t worry about learning them right now, they are beyond the scope of this course. The learning curve for using them is high, so we recommend not writing a code on them as you are still learning, but they are fun to use down the line and good to flex on your friends with.

To open **nano**, in your terminal window, run:

`nano`

To open a file with nano, in your terminal widow, run:

`nano filename.extension`

To open **Vim**, in your terminal window, run:

`vim`

Similarly, to open a file with Vim, run:

`vim filename.extension`

**Customizing the terminal**

On Mac, you can change the terminal’s font, color, etc by going to Settings -> Preferences. Make your changes, and simply close the customizer window to save.

On Windows, navigate to Settings, where you can make changes directly into the json file. For instructions on how to customize a json file, select this link [here (Links to an external site.)](https://nerdschalk.com/how-to-change-color-in-windows-terminal/#%3A%7E%3Atext%3DWindows%20Terminal%20comes%20with%20its%2Cicon%20and%20click%20%27Settings%27).

**List of Common Commands**

Here’s an appendix of commonly used commands.

**Commands**

**>**

$ cat oceans.txt > continents.txt

> takes the standard output of the command on the left and redirects it to the file on the right.

**>>**

$ cat glaciers.txt >> rivers.txt

>> takes the standard output of the command on the left and *appends* (adds) it to the file on the right.

**<**

$ cat < lakes.txt

< takes the standard input from the file on the right and inputs it into the program on the left.

**|**

$ cat volcanoes.txt | wc

| is a “pipe”. The | takes the standard output of the command on the left, and *pipes* it as standard input to the command on the right. You can think of this as “command to command” redirection.

**~/.bash\_profile**

$ nano ~/.bash\_profile

**~/.bash\_profile** is the name of file used to store environment settings. It is commonly called the “bash profile”. When a session starts, it will load the contents of the bash profile before executing commands.

**alias**

alias pd="pwd"

The alias command allows you to create keyboard shortcuts, or aliases, for commonly used commands.

**cd**

$ cd Desktop/

cd takes a directory name as an argument, and switches into that directory.

$ cd jan/memory

To navigate directly to a directory, use cd with the directory’s path as an argument. Here, cd jan/memory/ command navigates directly to the **jan/memory** directory.

**cd ..**

$ cd ..

To move up one directory, use cd ... Here, cd .. navigates up from **jan/memory/** to **jan/**.

**cp**

$ cp ada\_lovelace.txt historical/

cp copies files or directories. Here, we copy the file **ada\_lovelace.txt** and place it in the **historical/** directory

**Wildcards (\*)**

$ cp \* satire/

The wildcard \* selects all of the files in the current directory. The above example will copy all of the files in the current directory to the directory called **satire**. There are other types of wildcards, too, which are beyond the scope of this glossary.

$ cp m\*.txt scifi/

Here, m\*.txt selects all files in the working directory starting with “m” and ending with “.txt”, and copies them to scifi/.

**env**

$ env

The env command stands for “environment”, and returns a list of the environment variables for the current user.

**env | grep VARIABLE**

$ env | grep PATH

env | grep PATH is a command that displays the value of a single environment variable.

**export**

export USER="Jane Doe"

export makes the variable to be available to all child sessions initiated from the session you are in. This is a way to make the variable persist across programs.

**grep**

$ grep "Mount" mountains.txt

grep stands for “global regular expression print”. It searches files for lines that match a pattern and returns the results. It is case sensitive.

**grep -i**

$ grep -i "Mount" mountains.txt

grep -i enables the command to be case insensitive.

**grep -R**

$ grep -R Arctic /home/ccuser/workspace/geography

grep -R searches all files in a directory and outputs filenames and lines containing matched results. -R stands for “recursive”.

**grep -Rl**

$ grep -Rl Arctic /home/ccuser/workspace/geography

grep -Rl searches all files in a directory and outputs only filenames with matched results. -R stands for “recursive” and l stands for “files with matches”.

**HOME**

$ echo $HOME

The HOME variable is an environment variable that displays the path of the home directory.

**ls**

$ ls  
2014  2015  hardware.txt

ls lists all files and directories in the working directory

**ls -a**

$ ls -a  
.  ..  .preferences  action  drama comedy  genres.xt

ls -a lists all contents in the working directory, including hidden files and directories

**ls -l**

$ ls -l  
drwxr-xr-x 5  cc  eng  4096 Jun 24 16:51  action  
drwxr-xr-x 4  cc  eng  4096 Jun 24 16:51  comedy  
drwxr-xr-x 6  cc  eng  4096 Jun 24 16:51  drama  
-rw-r--r-- 1  cc  eng     0 Jun 24 16:51  genres.txt

ls -l lists all contents of a directory in long format. [Here’s what each column means](https://www.codecademy.com/courses/learn-the-command-line/lessons/command-line-manipulation/exercises/ls-l).

**ls -t**

ls -t orders files and directories by the time they were last modified.

**mkdir**

$ mkdir media

mkdir takes in a directory name as an argument, and then creates a new directory in the current working directory. Here we used mkdir to create a new directory named **media/**.

**mv**

$ mv superman.txt superhero/

To move a file into a directory, use mv with the source file as the first argument and the destination directory as the second argument. Here we move superman.txt into superhero/.

**nano**

$ nano hello.txt

*nano* is a command line text editor. It works just like a desktop text editor like TextEdit or Notepad, except that it is accessible from the command line and only accepts keyboard input.

**PATH**

$ echo $PATH  
   
/home/ccuser/.gem/ruby/2.0.0/bin:/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/bin

PATH is an environment variable that stores a list of directories separated by a colon. Each directory contains scripts for the command line to execute. PATH lists which directories contain scripts.

**pwd**

$ pwd  
/home/ccuser/workspace/blog

pwd prints the name of the working directory

**rm**

$ rm waterboy.txt

rm deletes files. Here we remove the file waterboy.txt from the file system.

**rm -r**

$ rm -r comedy

rm -r deletes a directory and all of its child directories.

**sed**

$ sed 's/snow/rain/' forests.txt

sed stands for “stream editor”. It accepts standard input and modifies it based on an *expression*, before displaying it as output data.

In the expression 's/snow/rain/':

* s: stands for “substitution”.
* snow: the search string, the text to find.
* rain: the replacement string, the text to add in place.

**sort**

$ sort lakes.txt

sort takes a filename or standard input and orders each line alphabetically, printing it to standard output.

**standard error**

*standard error*, abbreviated as stderr, is an error message outputted by a failed process.

**source**

source ~/.bash\_profile

source activates the changes in **~/.bash\_profile** for the current session. Instead of closing the terminal and needing to start a new session, source makes the changes available right away in the session we are in.

**standard input**

*standard input*, abbreviated as stdin, is information inputted into the terminal through the keyboard or input device.

**standard output**

*standard output*, abbreviated as stdout, is the information outputted after a process is run.

**touch**

$ touch data.txt

touch creates a new file inside the working directory. It takes in a file name as an argument, and then creates a new empty file in the current working directory. Here we used touch to create a new file named keyboard.txt inside the 2014/dec/ directory.

If the file exists, touch is used to update the modification time of the file

**uniq**

$ uniq lakes.txt

uniq, short for “unique”, takes a filename or standard input and prints out every line, removing any exact duplicates.