Command Line

* Basic User Commands
  + You can interact with your computer’s command line interface via the **terminal** (Mac) or **Command Prompt** (Windows) shell
  + Paths and Directories
    - When we want to interact with our files via the CLI, there are two types of paths we can choose:
      * Absolute Path: A path that starts at your home directory (also called the **root directory**) and lists all the folders and subfolders along the way.
        + For example, let’s say we want to find our resume. An absolute path might show something like this: ~/Documents/Resumes/My\_Current\_CV/CV.doc. The ~ at the beginning of this path tells the computer to build the path starting at your home director. Think of the ~ symbol as shorthand for your home directory, which will contain all of your files and documents.
      * Relative Path: A path that starts from your current position in the folder structure, referencing a given file or folder based on the starting position
        + For example, if you’re currently in the Documents directory, then the relative path to our resume would be: Resumes/My\_Current\_CV/CV.doc. This path does not start at your home directory (~)! It starts at your current position in the file system, which is Documents in this example.
      * Pro Tip: When you’re navigating in the command line, ./ is shorthand for “start from the current working directory,” whereas ~/ is shorthand for “start from the home directory.” If you type cd~ in the terminal, it will take you to your home directory.
  + Commands
    - Common CLI commands we’ll be using to navigate and manipulate our files:
      * Directory Contents: ls
        + When you open up your CLI, one thing that can be helpful to do is list all of the files and folders in your current directory, to help remember where you are.
        + To do this, we would use: ls
        + For example, to list all of the files and folders in your home directory, you would type:

ls ~

* + - * Command: Print Working Directory pwd
        + When you first start using the CLI, there are no clear visual guides to help you remember the path to your files and folders or where you currently are within your file system. That’s when the following command comes in handy
        + If you are working in a directory and want to see the absolute path to your current location, you would use: pwd
        + This stands for “print working directory.” In programing, we’ll use the word **print** to mean “show me the results of the command I just typed.”
      * Command: Change Directory cd
        + To change directories, we’ll use the cd command
        + In our CLI, any commands we type can be strung together. So for instance, we can tell it to change the directory in a specific order: to move up a folder level, down a folder level, or to some other location entirely
        + To move up a folder level, we would use: cd ..

Those two periods tell our CLI to go up one directory

* + - * Command: Make Directory mkdir
        + Let’s say we want to make a new folder from our desktop. First, navigate to your desktop (cd desktop/). Next, we would use the command mkdir to “make a new directory.”
      * Command: Create File (touch)
        + What if you want to create a new file? You can create new, empty files within the terminal using touch
        + For example, if you want to create an empty text file named blank.txt, you can simply type: touch blank.txt
      * Command: Copy File (cp)
        + The next few commands we’ll learn require additional **arguments**. IN other words, they’ll require you to type both the command and the action you want it to take. For example, when we typed mkdir we then had to tell it the name of the folder we wanted to make
        + **Copy / Paste**

Say we want to copy a file on our desktop. Using our mouse and a GUI, we might click a menu or use a keyboard shortcut like ctrl-c, after which we would manually navigate to the new location, then click or use ctrl-v to paste the file... How tedious!

With the CLI, we can speed this process up a lot by simplying using the command cp. You can think of this command as combining “copy” and “paste” together

To use this command and copy/paste a file in the CLI, you would type cp [original filename][location for duplicate file]

Let's give this a shot. We want to copy that blank.txt file on our desktops and place into our general\_assembly folder. What would we do?

cp ~/desktop/blank.txt ~/desktop/general\_assembly

Pro Tip: In the CLI, we separate arguments with a single space, just like typing a sentence!

* + - * Command: Remove File (rm)
        + Oops! That file turned out to be a mistake. Now what?
        + Instead of clicking a menu or dragging a file to the trash, we can use the CLI to delete files too!
        + First, jump into the general\_assembly folder from your desktop:

cd general\_assembly

* + - * + Then, to remove the file blank.txt , we would type:

rm blank.txt

* + - * + Now wait, is it really that simple? Yes... but be careful! This command is not like sending a file to the trash; you can't get it back once you've deleted it. Once you type rm [file], there is no recovering that file. So be mindful when using this command!
        + Try this yourself, but make sure to try it on a blank dummy file that you don't mind losing!!
      * Command: Move File (mv)
        + If you don't want to copy a file and paste it in another directory, you can just move the file from one directory to another using the mv command.
        + Similar to cp, this command needs us to tell it which file to move and where to put it. And again, you can use either relative or absolute paths!
        + To use this command, you would type: mv [original filename] [location to send file].
        + Let's try it with blank.txt. First, navigate back to your desktop. Then move the file into your general\_assembly folder.
        + mv blank.txt ./general\_assembly
      * Command Flags
        + Earlier, we taught you about multiple arguments, where a command requires additional information before it can execute properly. To speed up our workflow on the CLI, we can use additional arguments called flags to modify our commands further.
        + Flags are typed after a command, like this ls [flag] and take the form of a hyphen and letter. Some common flags we'll be using include: -r, -f, -v, -a, and -l.
        + Flags can be used to accomplish multiple types of tasks, depending on what command they are modifying.
        + For example, let's say we wanted to list the contents of our general\_assembly folder; we'd use the ls command. But what if we wanted to list items in reverse order? We would add a flag!

ls -r general\_assembly

* + - * + To list all the contents of our general\_assembly directory, even hidden files, we might type:

ls -a general\_assembly

* + - * + To see a list of all possible flags and their definitions you can look at a command's "manual page" (a.k.a. man page). For the command ls we would do this with the following command

man ls

* + - * + Once you are in the "man page," use the arrow keys to navigate and hit q to quit and go back to your terminal.
        + Pro tip: Flags can even be combined! However, proceed with caution when doing so; always experiment on blank files first.