**Intro to Bash**

**Unix shell basics**

* Unix/Linux: operating systems (OS) that manage computer hardware and provide services for running applications
* Operating system: software that manages all the hardware and software on a computer, and provides a way for users and applications to interact with the hardware
  + The manager of a computer, coordinating hardware, running applications, and providing a way for users to interact with the system
  + Examples: Windows, macOS (Unix-based), Linux
* Shell: a program that lets you interact with the OS using text commands
  + It’s the interface between the user and the OS
  + Command interpreter and mini programming environment
* Unix/Linux shell: program that provides a command line interface (CLI) to interact with the operating system
  + Environment where you can type text commands and have the system execute them, rather than using a graphical user interface like Finder of Windows Explorer
  + Different types of shells: Bsh, Zsh, Sh, etc.
* Flow: user 🡪 shell 🡪 operating system 🡪 hardware
* The Unix/Linux shell describes a program that takes commands from an input, like the keyboard, and passes them to an operating system that will execute them.
* In contrast, a graphical user interface (GUI), the shell simultaneously acts a command line interface and a programming language that allows you to perform tasks on your operating system.

**To find terminal on a mac:**

* Demonstrate opening terminal

**Why interact with system through the shell rather than a GUI?**

* Can quickly and easily navigate through directories on your computer, make, copy and search for files in a systematic way, and construct pipelines that will execute complex tasks on large datasets.

**Why learn to use a shell?**

* GUIs enable you to interact with your files and software in a very limited way, usually through point and click options that tell the software how to run.
* Again, these options are very limited, and often do not represent the full potential of the software.
* The options in the GUI may not be optimal for your dataset, and you may run into a “bug” where the GUI crashes or times out.
* Shell scripting allows for fluid transitions between programming languages, directories and projects, all of which can be accessed via the terminal.
* The terminal is also a great environment for file management.
* So, in summary, some advantages of using the shell are:
  + All possible software options are available
  + Software versions are easy to update
  + Debugging is easier

**The Bash shell**

**The absolute basics of the bash shell**

* BASH: the Bourne Again Shell – Unix/Linux command-line shell and scripting environment/language.
  + Allows users to control their Unix/Linux system efficiently and automate repetitive tasks
* There are different types of Unix shells, but the most popular is Bash, or the Bourne Again Shell.
* Most of us will be using Bash, and Dartmouth’s High Performance Computing (HPC) system uses Bash, so that’s what we’ll introduce this afternoon.

Open terminal

* When we open terminal, we can enter commands, which are typically composed of three components.
  + The name of the command
  + Any flags or options you want to run with this command, but these are not always required
  + A file or directory to act on
* Run this code:
  + cd /Users/ashleekorsberg/Ash\'s\ Documents/bash\_session

mkdir -p ./test\_intro\_to\_r/code

* Note: control + c to return to shell prompt
  + Description of code
    - mkdir: short for “make directory”, so we’re telling the OS to create a folder
    - -p: an option that lets us make parent and subdirectories at the same time
    - Last argument: the names of the directory and subdirectory that we want to make
* There are also commands to list files in our current working directory, to change the current directory, and to find our current directory
* Run this code:
  + ls

pwd

cd

**Where is your data stored?**

* Run this code:
  + cd /Users/ashleekorsberg/Ash\'s\ Documents/project5/code/
  + Description of code
    - Each directory and subdirectory in the path are separated by the forward slash

**Absolute vs. relative paths**

* The command “pwd” returns the absolute path to your current working directory
  + The absolute path will list all of the directories to get from your current location to the root or home directory
  + The absolute path can get very long, especially if you have a lot of directories
* Relative paths make navigating the command line a little easier; you can think of relative paths as shortcuts
  + Relative paths allow us to move up or down through directories using shortcuts
  + A common shortcut is the two dots, which will move us one directory above our current location
* Run this code:
  + cd ..

pwd

* Description of code
  + We can see that now I’m one step “above” the code folder, in project5 folder
* This shortcut saves a lot of time, but to use these shortcuts, you have to have a good understanding of your directory structure.
* By default, your terminal will start your current directory as your home directory.
  + No matter where you are, you can always return to your home directory using the tilde and cd command.
* Run this code:
  + cd ~

**Logging on to Dartmouth’s Discovery Cluster**

* Most bioinformatics data analysis will require a lot of memory and computing power, and more than most laptops can efficiently handle.
* For these analyses, using a high performance computing cluster is often necessary.
  + A cluster is a collection of compute resources, called nodes, that are accessed remotely through your local machine.
  + You can use these resources for both data storge and data processing.
* The Discovery Cluster is a resource hosted by Dartmouth’s research computing team
* We’ll practice logging on to the Discovery cluster now. We’ll use a secure shell command, called ssh, to log on.
  + Very important to note that if you’re not on eduroam, you’ll need to be on VPN to access this system.
* Run this code:
  + ssh [f006d8f@discovery8.dartmouth.edu](mailto:f006d8f@discovery8.dartmouth.edu)

**Customize your environment (do this in Discovery cluster)**

* The command line environment describes a collection of variables that have been defined for you.
  + These are referred to as environment variables.
  + The “env” command will show you all environment variables available in the current shell.
* Run this code:
  + env
* One important environment variable is “HOME”, which contains the path to your home directory.
* You can print the definition of a variable with the “echo” command, and the “$” indicates that you’re referencing a variable name.
* Run this code:
  + env $HOME
* You can also define your own environmental variables.
* Run this code:
  + GENOME=”hg38.patch13”

echo $GENOME

* Note that we’re using all caps to define variable names. This isn’t required, but it’s standard naming convention in bash.
* Another useful bash command is “ls -lah”, which will show us all of our files in a list format, including all hidden files, and will include file sizes in human readable format.
* Run this code:
  + ls -lah
* Variables created during a remote session will not persist between sessions, unless the variable is saved as an environment file.
  + These are a set of files that are executed every time you start a new bash session.
  + These files are typically hidden, so we need to use the “ls” with “a” option to see them.
  + The “.bash\_profile” is an example of a hidden environment file. We can view the contents of the file with the “cat” command.
* Hidden files: commonly used for configuration files, system files and reducing clutter.
* Run this code:
  + cd ~

ls -a

cat .bash\_profile

* This file is run every time you start a bash session and contains variables used to configure the bash environment. Defining a variable in this file will enable the variable to persist between remote sessions. See an example of how to do this below.
* That’s all we’ll do in the HPC environment today.
* You’ll meet the HPC team next Monday afternoon, and learn a lot more about Dartmouth’s high performance computing system during that session. They’ll be able to answer all of your questions.
* Let’s exit the HPC system and go back to using terminal on our local computers.
  + Note: to get out of secure connection environment, type “exit”

**Data exploration**

* We can keep all of our work organized by creating project directories, so let’s create a directory for bootcamp called “QBSBootcamp” on our desktop.
  + You may notice that a lot of my directory names have no spaces. This is normal naming convention in bash.
  + The space is a special character, and special characters need to be escaped with the back slash.
  + To keep filenames neat and easy to type in the command line, most programmers will not use spaces in directory or file names, but rather use underscores, periods or hyphens in their directory and file names.
* Run this code:

*# Navigate to your desktop*

*cd /Users/ashleekorsberg/Desktop*

*# Make the directory.*

*mkdir -p qbs\_bootcamp*

*# Change to the newly-created directory.*

*cd qbs\_bootcamp*

*# Create a variable so we can get here quickly*

*QBS="/Users/ashleekorsberg/Desktop/qbs\_bootcamp"*

*###############*

*# Add variable definition to .bash\_profile*

*nano ~/.bash\_profile*

*# You may also need to provide yourself permission using sudo:*

*sudo nano ~/.bash\_profile*

*# copy and paste the variable definition above*

*# save .bash\_profile file (control+X, Y, enter)*

*###############*

*# Check your location on your machine*

*pwd*

*# List the contents of your directory*

*ls*

* As expected, our new directory is empty, so let’s add something to it. Please download the “sample\_data\_1.csv” file and save it to your Desktop, but don’t put it in the directory that we just created.
* So, we can copy entire files from one directory to another using the “cp” command, which is short for copy.
  + Let’s copy that CSV file from our Desktop into our QBS Bootcamp directory.
* Run this code:

*# copy file*

*cp /Users/ashleekorsberg/Desktop/sample\_data\_1.csv $QBS*

*# navigate to QBS bootcamp directory*

*cd $QBS*

*# list files in directory*

*ls*

* Then, if we navigate back to our QBS Bootcamp folder and list the files, we should see the CSV file.

**Viewing the contents of a file**

* The shell provides us with commands to view the contents of files.
  + For example, the “cat” command will print the contents of entire files. This is maybe not the desired behavior for larger files though.
* When working with larger files, which is common in our field, you probably don’t want to see the entire file contents displayed in terminal.
  + So, there are other commands that allow us to explore files with more control, and we’ll look at a couple examples of that now.
* Run this code:

*cat sample\_data\_1.csv*

*# Show the first 20 lines of the file*

*head -n 20 sample\_data\_1.csv*

*# Show the last 50 lines of the the file*

*tail -n 50 sample\_data\_1.csv*

*# Use the word count (wc) command with the lines option (-l) to show how many lines (rows) are in the dataset*

*wc -l sample\_data\_1.csv*

**Renaming and removing files**

* Sometimes, you may want to reorganize your directories, or rename a file, or move a file, all of which can be done with the “mv” command, which is short for move.
* Let’s copy the CSV file from the QBS Bootcamp directory to our home directory.
* Run this code:

*# Copy the file to your home directory*

*cp sample\_data\_1.csv ~/sample\_data\_1.csv*

* Now, let’s rename the copy of the CSV file that we just created.
* Run this code:

*# Rename the copied file*

*mv ~/sample\_data\_1.csv ~/sample\_data\_1\_copy.csv*

* We can also use the “mv” command to move a file to a new location. Let’s move the copy of the CSV file from the home directory back into the QBS bootcamp directory.
* Run this code:

*# Move the copy of the CSV file into your qbs\_bootcamp directory.*

*mv ~/sample\_data\_1\_copy.csv $QBS/sample\_data\_1\_copy.csv*

*#check the contents of your directory*

*ls*

* This was just an example to show you how to rename, move and copy files.
* Let’s remove the copy of the file with the “rm” command.
* Run the code in the Rmd file.

*# Remove the file*

*rm sample\_data\_1\_copy.csv*

*# Check files in directory*

*ls*

* Just a note to be careful about deleting files if you’re ever working in shared directories.
  + Because there’s no “Trash” equivalent in the shell.

**Manipulating file contents**

* There are other commands that allow you to manipulate and subset files based on specific filters or parameters.
* We can use the “cut” command to look at specific columns and rows in our dataset.
* Run this code:

*# Look at second column*

*cut -d',' -f2 sample\_data\_1.csv*

* But, if we don’t want to see all 100 rows of our dataset, we could combine the “cut” command with the “head” command using a “pipe”. Pipes send the output of the initial command (on the left) to the next command (on the right) with a single line of code.
* Run this code:

*# List only the first 20 lines*

*cut -d',' -f 2,4,5 sample\_data\_1.csv | head -n 20*

* We could also use the redirect operator to send the output of the “cut” command to create a new file that contains this information.
* Run this code:

*# Save contents to new file*

*cut -d',' -f 2,4,5 sample\_data\_1.csv | head -n 20 > sample\_data\_subset.csv*

*# look at head of this new file*

*head sample\_data\_subset.csv*

**Pattern matching with grep**

* The last topic we’ll talk about today is pattern matching, which can be helpful for exploring, filtering, sub setting and analyzing data.
* We can use the “grep” command to search for specific strings in the data file. “grep” is a pattern recognition tool that searches in files for character strings or parts of character strings.
* Run this code:

*# Search for the word "Medium"*

*grep "Medium" sample\_data\_1.csv*

* We can also combine regular characters with special characters to search for partial character strings. Some commonly used special characters are shown below.
* These are called regular expressions, and can be used with any of the things we’ve learned so far.
* Run this code:

*# List all files that end in .txt*

*ls \* .csv*

* You’ll probably see “grep” and regular expressions in some of your other classes.