Intro to Bash

QBS Bootcamp 2025

Unix Shell Basics

The Unix/Linux 'Shell' describes a program that takes commands from an input (eg. your keyboard) and passes them to an operating system that will execute them. In contrast to a Graphical User Interface (GUI), the Shell is simultaneously a command line interface (CLI) and a programming language that allows you to perform tasks on your system.

Do you know how to access the terminal window on your computer?

To find the terminal on a Mac:

1. Navigate to the "Go" drop down in the upper left hand corner.
2. Click on "Utilities". A window displaying your "Utilities" applications will pop up.
3. Double click on the "Terminal" application and your terminal will appear!

To find terminal on a Windows system:

- You can use an application such as MobaXterm or PuTTY.

Interacting with a system through the Shell has many advantages over a GUI. The Shell allows you to quickly and easily navigate through directories on your computer, make, copy and search files in a systematic way, and construct pipelines that will execute complex tasks on big datasets.

Importantly, the Shell allows us to do each of these in the context of Bioinformatics, and Bioinformatics software.

Why Learn to Use A Shell

GUIs enable you to interact with your files and software in very limited ways by clicking buttons or selecting check boxes that correspond to choices you can make about how the software can run. Due to the design of the GUI the options available for any piece of software are limited to the most popular options, but these options do not represent the full potential of the software used by the GUI wrapper. These options may not be optimal for the dataset that you are working with, and the software might not be the latest version. When this happens you will run into what we call "bugs" where the GUI crashes or times out before the data are processed. In this case you may be able to update your GUI, but often you're left looking desperately for another tool that will do something similar.

Shell scripting also allows for fluid transitions between programming languages, directories, and projects that can be accessed at the terminal. The terminal is a conducive environment for file management, a topic we will touch on in our class today.

Advantages of Using The Shell

- All possible software options are available
- Software versions are easy to update
- Debugging is easier

The Bash Shell

The Absolute Basics

There are different types of Unix shells, however the most popular is Bash (the Bourne Again Shell), which is also the most common on Linux systems. Since the majority of participants will be using the Bash shell, and this is the default shell used on Dartmouth's high performance computing system (which we will be using), this lesson will introduce the Shell through using the Bash shell, however most, if not all, content should be transferable to other Unix shells.

When you open your terminal application, you will be presented with the command prompt `$` when you are able to input commands. If the terminal is busy and cannot currently accept new commands, you will not be presented with the prompt.

When the prompt is shown, you can enter commands by typing them in after the prompt. Commands are typically composed of three components:

1. the name of the command itself
2. any flags or options you wish to run the command with (not always required)
3. a file or directory to act on (sometimes implicit)

```
mkdir -p ./test_intro_to_bash/code
```

In the above example, we are asking the Shell to pass the `mkdir` command to the operating system to create a folder. The `-p` option lets us make parent and subdirectories at the same time. Finally, we include an argument detailing the names of the directory and subdirectory we are making.

It's tricky to remember all of the command and what they do. If you ever get lost, you can access the manual pages for specific commands using the `man` command followed by the command for which you require information.

```
man mkdir
```

The shell provides us with commands that allow us to list files in our current working directory, change the current directory, and find our current directory.

```
# 'ls' command lists files in our current working directory
ls

# 'pwd' command displays the path of our current working directory
pwd

# 'cd' command allow us to change directories
cd /Users/Documents
```

Where is Your Data Stored?

I often organize each of my projects into their own directory (folder). Within that directory, I have several subdirectories such as "code", "figures", and "saved_objects". This, however, can pose a problem when I go to search for a directory on my computer called "code". In some cases there may be multiple directories called "code" and we would need to distinguish which directory contains the code of interest. We need to feed the specific "address" of the code folder we want to the command line. Let's say I want to get to the directory with the "code" subdirectory for project 5 that is located on my desktop. I might employ the following command:

```
cd /Users/Documents/project5/code/
```

The path `/Users/Documents/project5/code/` refers specifically to the code directory in project5. Each directory and sub-directory in the path are separated by the forward slash `/` to indicate the path through the directories to the directory of interest. You will need to submit the location of the files you would like to analyze using a path when you run bioinformatic software.

Absolute vs. Relative Paths

The command `pwd`returns the absolute path to your current working directory, the list of all directories and subdirectories to the get from the current directory to the root or home directory. You can see that absolute paths can get long and unwieldy, especially if you have very detailed or long directory names.

One "shortcut" that makes navigating the command line a bit easier is using a relative path. A relative path uses the directory structure (which we can see in our absolute path returned by the command `pwd`) to move up or down through directories using shortcuts. One very common shortcut is `..` which translates to the directory one level "above" your current directory.

Using the example from `/Users/Documents/project5/code/` we could get back to the `project5` directory using the absolute path command `cd /Users/f002yt8/Documents/project5/` or we could use the relative path with the command `cd ..`

This shortcut saves a lot of time and typing BUT it requires that you have a good understanding of where you are in your working directory structure, so do not be shy about using the pwd command.

```
# ".." tells shell to move your current directory up one subdirectory
cd ..

# check working directory again
pwd
```

Relative paths are contrasted to absolute paths which always starts with a '/' and will start at the root (highest level) of the directory tree, and work from wherever you are in the directory substructure. A key difference to remember is the absolute path will always point you to the same location, regardless of your current working directory. A relative path like `cd ../` will always point to the directory above your current working directory, but your new location will be relative to your current working directory.

By default, your terminal application will start your current directory as your home directory (more on that later). No matter where you are, you can always get back to your home directory using the tilde `~` with the `cd` command.

```
cd ~
```

Logging on to Dartmouth's Discovery Cluster

Most NGS (next generation sequencing) data analysis will require a lot of memory and computing power, more than most laptops can handle efficiently. For these analyses, using a high performance compute (HPC) cluster is often necessary. A cluster is a collection of compute resources, called nodes, that are accessed remotely through your local machine. You can leverage these resources for both data storage and data processing. These compute resources work together as a single system.

The discovery cluster is a resource hosted by Dartmouth's Research Computing team. Let's log onto the discovery cluster now. We will use a secure shell command `ssh` to log onto the discovery cluster (if you are not on campus you will need to be on the VPN network to log on to the cluster).

```
# Establish the secure shell connection ****REPLACE netID WITH YOUR OWN ID****
ssh netID@discovery8.dartmouth.edu

# Enter your password at the prompt (when you type no characters will show up to preserve privacy)
netID@discovery8.dartmouth.edu's password:

# Successful log in!
(base) [netID@discovery8 ~]$
```

Customize Your Environment

The command line environment describes a collection of variables that have been defined for you to provide context for the commands that you run. These are referred to as environment variables. The `env` command will show all environment variables available in the current shell. Try that now:

```
env
```

One important environment variable is `$HOME`, which contains the path to your home directory. You can print the definition of a variable with the `echo` command, the preceding `$` indicates you're referencing a variable. For example:

```
echo $HOME
```

You can define your own environmental variables during a remote session. These can be virtually anything. For example, perhaps you want to save the name of the genome version you are working with in your current session, so it can be easily called multiple times in some bash code you are writing.

```
# set the variable
GENOME="hg38.patch13"

# call it with echo and the $
echo $GENOME
```

Note: You will notice I'm using all caps for my variables, this isn't required but it is good practice to indicate to yourself that you're using a variable.

We might want run the command ls -lah command to show files in a list format, including all hidden files, and with file sizes in human readable format.

```
# return files
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 `ls` with the `-a` flag to see them. The `.bash_profile` is an example of an environment file, we can view the contents of this file with the `cat` command.

```
# navigate to your home directory
cd ~

# view files in current working directory and include hidden files
ls -a

# view contents of bash profile
cat .bash_profile
```

The `.bash_profile` is run every time you start a bash session and contains variables used to configure the bash environment. Defining a variable in the `.bash_profile` will enable the variable to persist between remote sessions. Lets define the variable `$LIST` in our `~/.bash_profile`.

```
# use the nano text editor to add the line ' LIST="ls" ' to your bash_profile
nano ~/.bash_profile

# You may also need to provide yourself permission using sudo:
sudo nano ~/.bash_profile

# source the new bash_profile to add the environment variables to your current session (or start a new
source ~/.bash_profile

$LIST
```

Now `$LIST` will be set as an environment variable every time we start a new bash session terminal.

That's enough for the HPC environment today. Dr. Darabos will be providing extensive instruction on using Discovery in a later Bootcamp session. For now, please close your terminal window and open a fresh new terminal so we can begin digging into some data!

Data Exploration

It is a good idea to stay organized when working on the terminal by creating project directories, so let's start by making a directory called `qbs_bootcamp` on our desktops.

You will notice that I chose a directory name with no spaces. The space is a special character, special characters need to be escaped with the `\` and so `qbs_bootcamp` would look like `QBS\Bootcamp` with the escape characters. You can see that file names with spaces become unwieldy to type out so most programmers will replace spaces with `_`, `.`, or `-` in their filenames to keep everything neat.

```
# 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, the new directory that you created is empty. Let's add something to it! Please download the CSV file included in the email we sent last night. Download it and put it on your Desktop, but not in the directory we just created (yet!).

We can copy entire files from one directory to another using a short command `cp`, short of copy. Let's copy the CSV file from our Desktop into the `qbs_bootcamp` directory we just created:

```
# copy file
cp /Users/ashleekorsberg/Desktop/sample_data_1.csv $QBS

# navigate to QBS bootcamp directory
cd $QBS

# list files in directory
ls
```

Viewing The Contents of A File

The shell provides us with commands to view the contents of files in definite ways. The `cat` command for example (which stands for concatenate) will print the entire contents of a file to the terminal. This can be useful for smaller files, but as you will see with larger files can quickly fill the terminal with more lines of data than it can display.

```
cat sample_data_1.csv
```

When working with larger files, which is common in bioinformatics, you may not wish to look at a portion of a file. Other commands exist that allow you to explore file contents with more control.

- more shows you as much of the file as can be shown in the size of the terminal screen you have open, and you can continue to "scroll" through the rest of the file by using the space bar
- less is a similar command to more, and has advantages such as not persisting in the terminal, and being searchable
- head will print the first 10 lines by default, but this number can be controlled with the -n option
- tail will print the final 10 lines of a file, and can also be controlled with the -n option

```
# 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 d
wc -l sample_data_1.csv
```

Renaming and Removing Files

Sometimes you will need to reorganize your directories or rename a file, which can be achieved with the `mv` command. Let's start by copying the CSV file from the `qbs_bootcamp` directory to your home directory.

```
# 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.

```
# Rename the copied file
mv ~/sample_data_1.csv ~/sample_data_1_copy.csv
```

You can also use the `mv` command to move a file to a new location. Let's move the copy of the CSV file from your home directory into your `qbs_bootcamp` directory.

```
# 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
```

Copying the CSV file was just an exercise to show you how the tools work. In practice, you will want to keep your directories as neat as possible as you accumulate a lot of files. Let's remove the copy of the CSV file with the rm command.

```
# For the sake of being careful, let's first list the details file to be removed
ls -l sample_data_1_copy.csv

# Remove the file
rm sample_data_1_copy.csv

# Check files in directory
ls
```

You will notice that before the file was deleted you were asked if you were sure you wanted this file deleted. You want to be careful not to remove files that you did not create if you are working in shared directories. If you want to bypass this checkpoint, you can use the `-f` flag with `rm -f` to force the removal of a file, but be careful with this, as there is no Trash equivalent in the shell.

Manipulating File Contents

Some commands enable you to manipulate and subset files based on specific parameters. One useful example is the cut command, which allows you to 'cut' a file based on the options you select. We could use `cut` to look at a certain column in the CSV file.

```
# Look at second column
cut -d',' -f2 sample_data_1.csv
```

To prevent all rows being printed to our console, we could combine the `cut` command with the `head` command using a 'pipe', specified by a '|'. Pipes send the output of the initial command (on the left) to the next command (on the right), with a single line of code.

```
# List only the first 20 lines
cut -d',' -f 2,4,5 sample_data_1.csv | head -n 20
```

Similarly to how we used the pipe operator (|) above, we could use the redirect operator(>) to send the output of the cut command to create a new counts file.

```
# 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

Often we will want to pull a specific piece of information from a large file. We can use the `grep` command to search for character strings in the data file.

```
# Search for the word "Medium"
grep "Medium" sample_data_1.csv
```

`grep` is a pattern recognition tool that searches in files for a character string we can define. We can define the entire character string, as we did above, or combine regular characters with special characters (or 'wildcards') to search for specific types of matches. Some commonly used special characters are included below.

- *: wildcard stands for any number of anything
- ^: start of the line
- $: end of line
- `[0-9 or \d]`: any number (0123456789)
- `[a-z]`: any lowercase letter
- `[A-Z]`: any uppercase letter
- `\t`: a tab
- `\s`: a space

These regular expressions can be used with any of the tools that you have learned thus far, so if we wanted to list all of the files in our directory that end in .txt we could use the following command.

```
# List all files that end in .txt
ls *.csv
```

We can even enhance the power of these regular expressions by specifying how many times we expect to see the regular expression with quantifiers.

- X*: 0 or more repetitions of X
- X+: 1 or more repetitions of X
- X?: 0 or 1 instances of X

You will see regular expressions in some of your other classes.