Command line basics

Credit: Heavily borrowed from Software Carpentry Foundation

Contents

Key Points

- A shell is a program whose primary purpose is to read commands and run other programs.
- This lesson uses Bash, the default shell in many implementations of Unix.
- Programs can be run in Bash by entering commands at the command-line prompt.
- The shell’s main advantages are its high action-to-keystroke ratio, its support for automating repetitive tasks, and its capacity to access networked machines.
- The shell’s main disadvantages are its primarily textual nature and how cryptic its commands and operation can be.

Graphical user interface vs. the Unix shell

Humans and computers commonly interact in many different ways, such as through a keyboard and mouse, touch screen interfaces, or using speech recognition systems. The most widely used way to interact with personal computers is called a **graphical user interface (GUI)**. With a GUI, we give instructions by clicking a mouse and using menu-driven interactions.

While the visual aid of a GUI makes it intuitive to learn, this way of delivering instructions to a computer scales very poorly. Imagine the following task: for a literature search, you have to copy the third line of one thousand text files in one thousand different directories and paste it into a single file. Using a GUI, you would not only be clicking at your desk for several hours, but you could potentially also commit an error in the process of completing this repetitive task. This is where we take advantage of **the Unix shell**.

What the hell is "the shell"?

The Unix shell is **both a command-line interface (CLI) and a scripting language*, allowing such repetitive tasks to be done automatically and fast. With the proper commands, the shell can repeat tasks with or without some modification as many times as we want. Using the shell, the task in the literature example can be accomplished in seconds.

It is a computer program that allows you to "talk" to your computer and give it tasks. But you need to speak its language!

In Unix-based machines, which includes Macs (they use Unix in the background with a fluffy interface for the user), the command line or shell is easy to access. On a Mac, you can do so by going to the app "Terminal".

On Windows: I've heard it's possible but I cannot attest to it. Last I used windows, it wasn't.

Now you know what it is!

Opening a terminal window

For this introductory activity, we will use the terminal pane you opened in our B216_S23 binder.

In your life, know that you can access a terminal in any computer:

- Mac or other Unix based systems: In your personal or lab computer, go to the "Terminal" application. Tip: if you don't know where to find it, open Finder and search for it.
- Windows: not ideal but possible. I don't know how to do it though. Good luck!

Why should I use the command line?

It may not seem worth your while to run a program ONCE using the command-line, but when you find yourself contemplating manually repeating a computational task 100 times, the appeal becomes clear!

- high action-to-keystroke ratio.
- support for automating repetitive tasks
- capacity to access networked machines.

In bioinformatics, there are often user friendly versions of software (though not always), but there is most certainly a command line version! Why?

- the use of memory is more efficient
- when dealing with huge files, the command line is the only way!

Let's get familiar with the terminal!

Remember, the terminal is a way for you to talk to your computer. Or, in this case, a virtual space.

Note: a directory is what you ordinarily call a "folder" in a computer. For example, on my mac I have a `Documents` folder where I have a `B216_S23` folder. Those are both directories, or subdirectories. The ultimate direcotry is the one that contains all the subdirectories! On Mac/Unix machines, that's given by `/`.

Checking where you are

Once you open a terminal, run the **"path to working directory"** command:

```
#copy the command below and paste it into your terminal. Or, better, type it for practice!
pwd
```

- Before we move further, open a regular text file in your computer.
- Name it so that it reflects your BMC/Haverford username. E.g. my BMC email is bbitarello@brynmawr.edu, so my filename would be `lab8_bbitarello_answer_sheet.txt`.
- All done! Now, whenever a question comes up, type your answers into that file.
- Upload that to Moodle when you're done.

Q1. What does this show you? (I.e., what is the output of the command above) Copy your output into your answer sheet.

Listing contents

Now check what's in that directory using the "list" command:

```
ls
```

Q2. What do you think the above command does?

Notice a resemblance between the output of your command line and the left sidebar panel? That's because the terminal is a different way to access things! Instead of clicking on things through a user interface, you type commands.

After you typed `ls` into your terminal, you saw something in your output. Some of those things are directories. Let's have a look by using the "change directory" command:

```
cd filestoshare/
```

You are now at a different subdirectory! Convince yourself by typing `pwd`:

```
pwd
```

Q3. What is the output of the command above? Copy your output into your answer sheet.

Now you can use `ls` again to see what's inside this subdirectory:

```
ls
```

Q4. What are the names of the subdirectories in this directory? Copy your output into your answer sheet.

Q5. How can you differentiate file names from directory names?

Changing directories

Now we want to "go back" to where we were before. We will use the `cd` command again.

First, check where you are by using the `pwd` command:

```
pwd
```

Great! Now let's go back where we were before:

```
cd ~
# ~ is a shortcut that makes you not have to type the entire path to your home folder
```

Check that it worked. You should now be at `/home/jovyan`. Note: if you were using your personal computer, say a Mac, this would be `/home/janedoe`, assuming your user name is janedoe. Up here in this workspace, all user names are the same: `jovian`.

Trivia: why do you think the user name in this virtual space is `jovian`?

Q6. How can you check that you are indeed there? What command would you use?

Looking at a file without opening it (!)

Now, let's demonstrate how you can access the contents of a file without "opening it".

Go back to the `filestoshare/fasta` directory.

Q7. What command do you use for this?

Now, once again use the `ls` command to see what's inside this subdirectory:

```
ls
```

Q8. How many files are in there?

Let's investigate the file called `sarscov2genome.fasta` by using the `head` command:

```
head sarscov2genome.fasta
```

The head command shows you the first 10 lines of a textfile.

Now let's use the tail command. Repeat the above command replacing `head` by `tail`:

```
#type your command in your terminal
```

Q9. What does this do?

Finally, you can also check how many lines a textfile has without ever having to open it. (this can prevent you from crashing your computer when you have a very large file…)

Use the "word count" command, `wc`:

```
wc sarscov2genome.fasta
```

Q10. What did this do?

Great! Now let's move up one directory. Up means towards the directory that contains the one you are in. Whenever you are in a subdirectory, you can go back to the one "above" it by simply typing `../`.

```
cd ../
```

Check that it worked by typing `pwd` once again:

```
#type your command in your terminal
```

Now move into the `fastq/gartersnake` directory. That's where we will work now.

```
cd fastq/gartersnake
```

Seeing the contents of a file bit by bit

Let's combine what we've learned.

First, list the contents in this directory.

Let's look at the `Female2-oral1.fastq` file.

Q11. How many lines does it have? Use the commands we learned. Copy your command and output to your answer sheet.

Hopefully that convinced you that you don't want to try and open that file! But what if you want to look at the contents bit by bit, instead of a fixed number of lines?

The `more` command!

We saw how `head` and `tail` work. Here is another useful one: the `more` command. Try it on `Female2-oral1.fastq`. It will show you the contents of the file until your terminal is full.

- Tap `enter` to keep seeing what comes next in the file.
- Congrats! You looked at a giant file without having to open it!

When you're done here, go to the left sidebar and open the file called `01_fastqe.html`.

Learn More

The Software Carpentry Foundation has great stuff: e.g. https://swcarpentry.github.io/shell-novice/01-intro/index.html

DataCamp has bash courses and they are not bad!

The `wc` command in Linux. https://www.howtogeek.com/812441/wc-command-in-linux/

The `pwd` command in Linux. https://phoenixnap.com/kb/pwd-linux