Shell genomics

2023-11-15

Table of contents

Preface

The content of this workshop is based on the material by the Data Carpentry: Introduction to the shell for genomics data (Becker et al. 2019).

Command line interface (CLI) and graphic user interface (GUI) are different ways of interacting with a computer's operating system. They have different pros and cons. Most people are familiar with the GUI as it is the default interface for most software, particularly on Windows and Mac OS. When using the GUI, you see visual representations of files, folders, applications etc. When using the CLI, you work largely with text representations of files, folders, input and output etc. The shell is a program that presents a command line interface that allows you to control your computer by typing instructions with a keyboard.

There are several reasons to learn how to use the CLI:

- For most bioinformatics tools, there are no graphical interfaces. If you want to work in metagenomics or genomics you're going to need to use the CLI/shell.
- The shell gives you power. The command line allows you to work more efficiently. Tasks that are repetitive (e.g. renaming hundreds of files) can be automated. Tasks that are tedious (e.g. testing a range of input parameters) can be simplified.
- To use remote computers or cloud computing, you need to use the shell.

Schedule

	Setup	Download files required for the lesson
00:00	1. Introducing the Shell	What is a command shell and why would I use one? How can I move around on my computer? How can I see what files and directories I have? How can I specify the location of a file or directory on my computer?
00:30	2. Navigating Files and Directories	How can I perform operations on files outside of my working directory? What are some navigational shortcuts I can use to make my work more efficient?

	Setup	Download files required for the lesson
01:20	3. Working with Files and Directories	How can I view and search file contents? How can I create, copy and delete files and directories? How can I control who has permission to modify a file? How can I repeat recently used commands?
02:05	4. Redirection	How can I search within files? How can I combine existing commands to do new things?
02:50	5. Writing Scripts and Working with Data	How can we automate a commonly used set of commands?
03:30	6. Project Organization	How can I organize my file system for a new bioinformatics project? How can I document my work?
04:00	Finish	

The actual schedule may vary slightly depending on the topics and exercises chosen by the instructor.

1 Introducing the shell

Time

- Teaching: 20 min
- Exercises: 10 min

Questions

- What is a command shell and why would I use one?
- How can I move around on my computer?
- How can I see what files and directories I have?
- How can I specify the location of a file or directory on my computer?

Objectives

- Describe key reasons for learning shell.
- Navigate your file system using the command line.
- Access and read help files for `bash` programs and use help files to identify useful command options.
- Demonstrate the use of tab completion, and explain its advantages.

1.1 What is a shell and why should I care?

A *shell* is a computer program that presents a command line interface which allows you to control your computer using commands entered with a keyboard instead of controlling graphical user interfaces (GUIs) with a mouse/keyboard/touchscreen combination.

There are many reasons to learn about the shell:

- Many bioinformatics tools can only be used through a command line interface. Many more have features and parameter options which are not available in the GUI. BLAST is an example. Many of the advanced functions are only accessible to users who know how to use a shell.

- The shell makes your work less boring. In bioinformatics you often need to repeat tasks with a large number of files. With the shell, you can automate those repetitive tasks and leave you free to do more exciting things.
- The shell makes your work less error-prone. When humans do the same thing a hundred different times (or even ten times), they're likely to make a mistake. Your computer can do the same thing a thousand times with no mistakes.
- The shell makes your work more reproducible. When you carry out your work in the command-line (rather than a GUI), your computer keeps a record of every step that you've carried out, which you can use to re-do your work when you need to. It also gives you a way to communicate unambiguously what you've done, so that others can inspect or apply your process to new data.
- Many bioinformatic tasks require large amounts of computing power and can't realistically be run on your own machine. These tasks are best performed using remote computers or cloud computing, which can only be accessed through a shell.

In this lesson you will learn how to use the command line interface to move around in your file system.

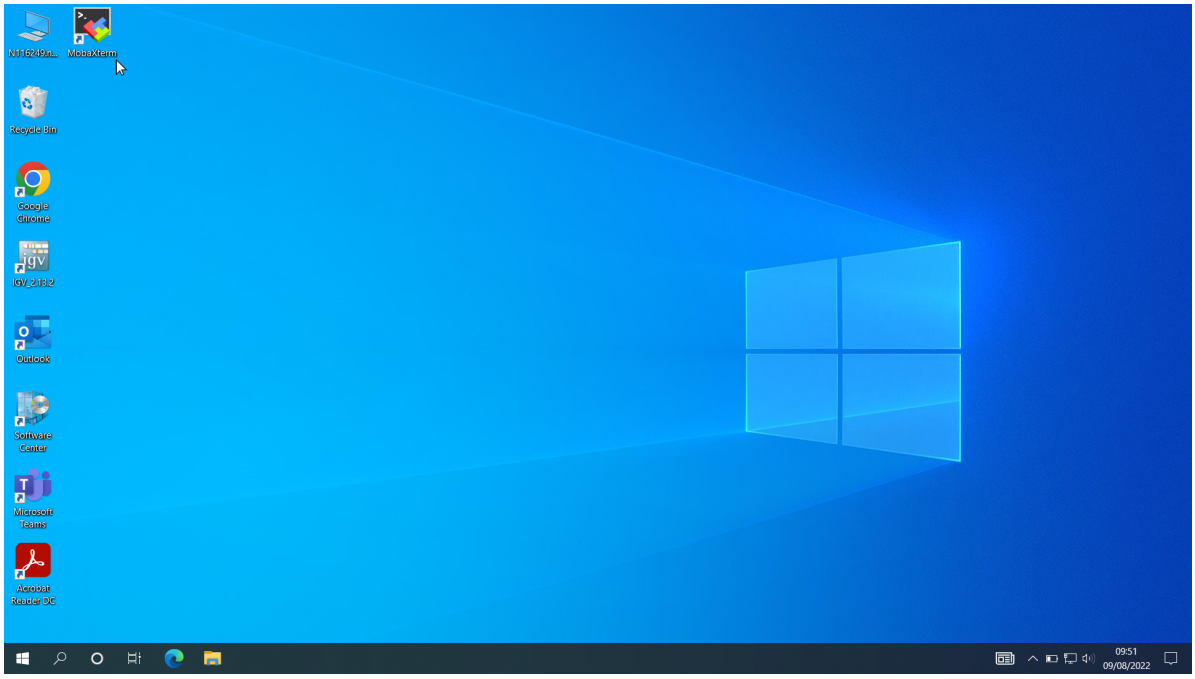
1.2 How to access the shell

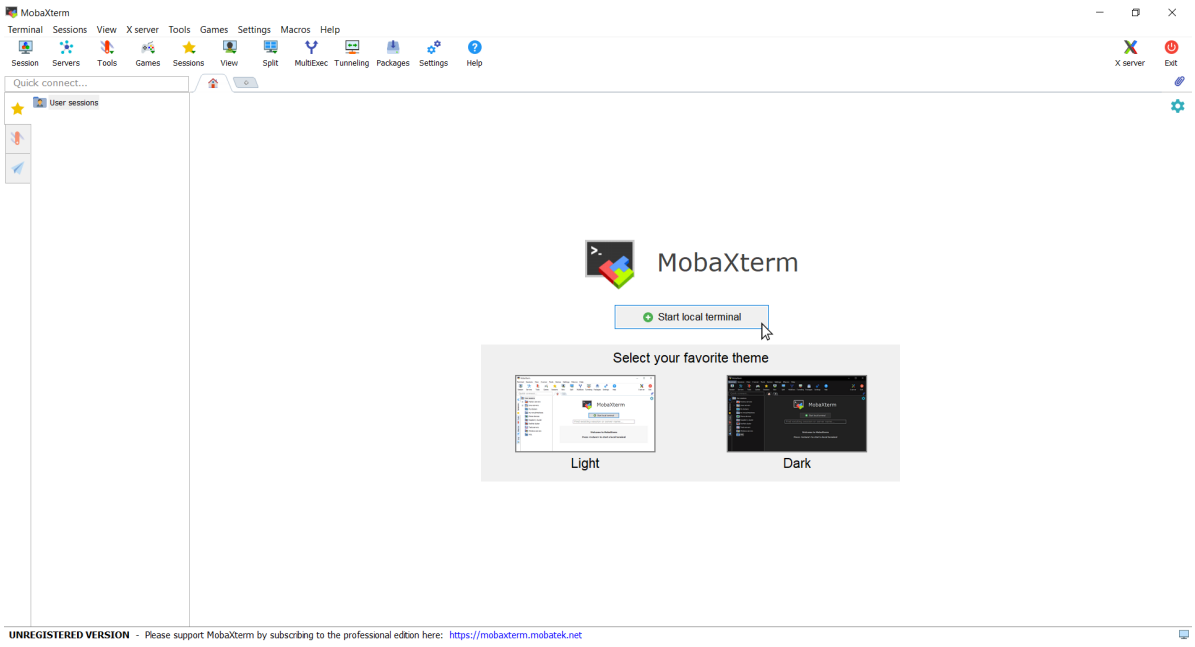
The Terminal is a window into which we will use to type commands in.

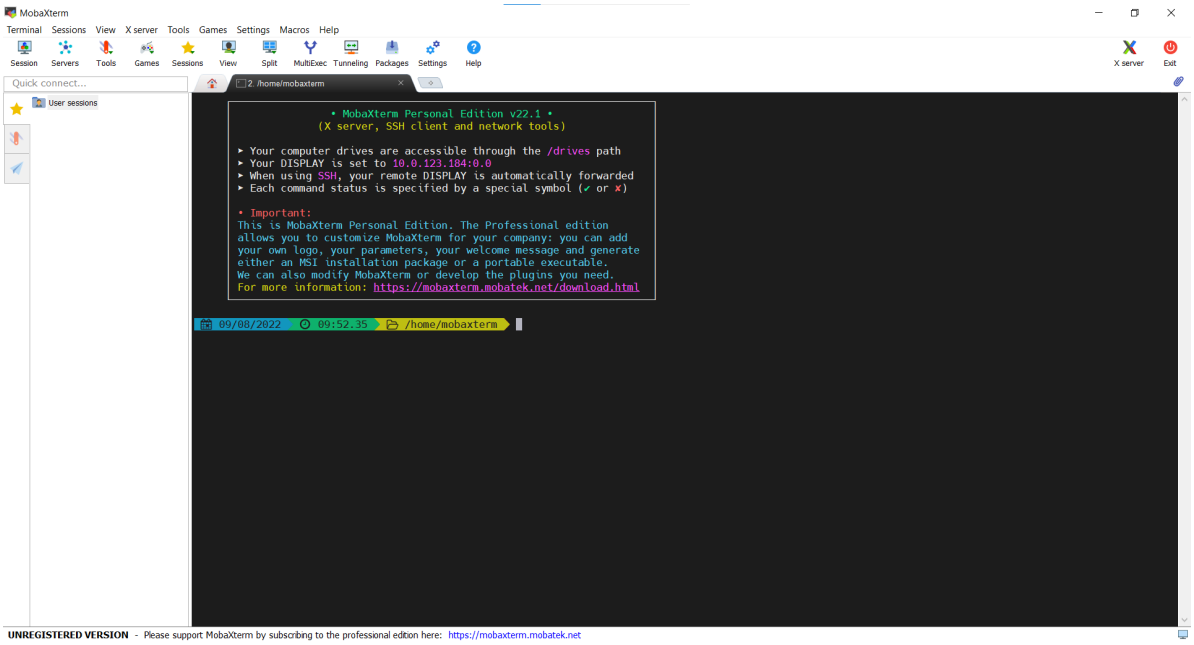
1.2.1 Windows

If you're using Windows, you need to use a separate program to access the shell. You can use MobaXterm, a terminal for Windows, that is already installed on your machine.

1. Click on the MobaXterm icon on your desktop.
2. Select `Start local terminal`

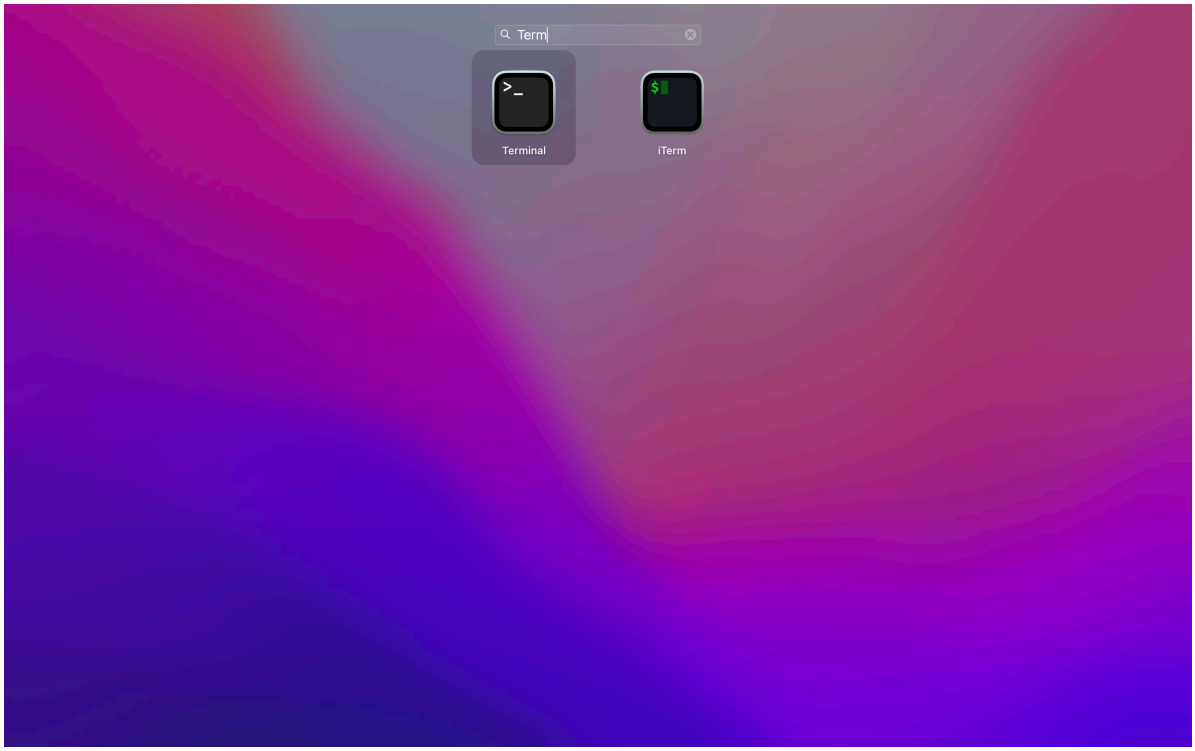

MobaXterm
Recycle Bin
Google Chrome
IGV_2.13.2
Outlook
Software Center
Microsoft Teams
Acrobat Reader DC
09:51
09/08/2022

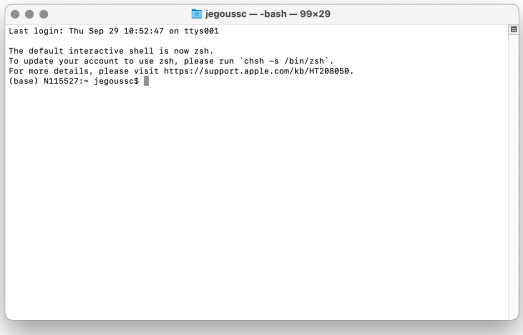

MobaXterm
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
User sessions
MobaXterm
Start local terminal
Select your favorite theme
Light
Dark
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net



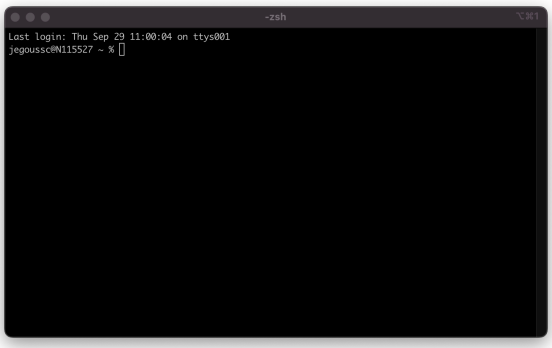
1.2.2 MacOS

On a Mac or Linux machine, you can access a shell through a program called “Terminal”, which is already available on your computer.





(a) Terminal

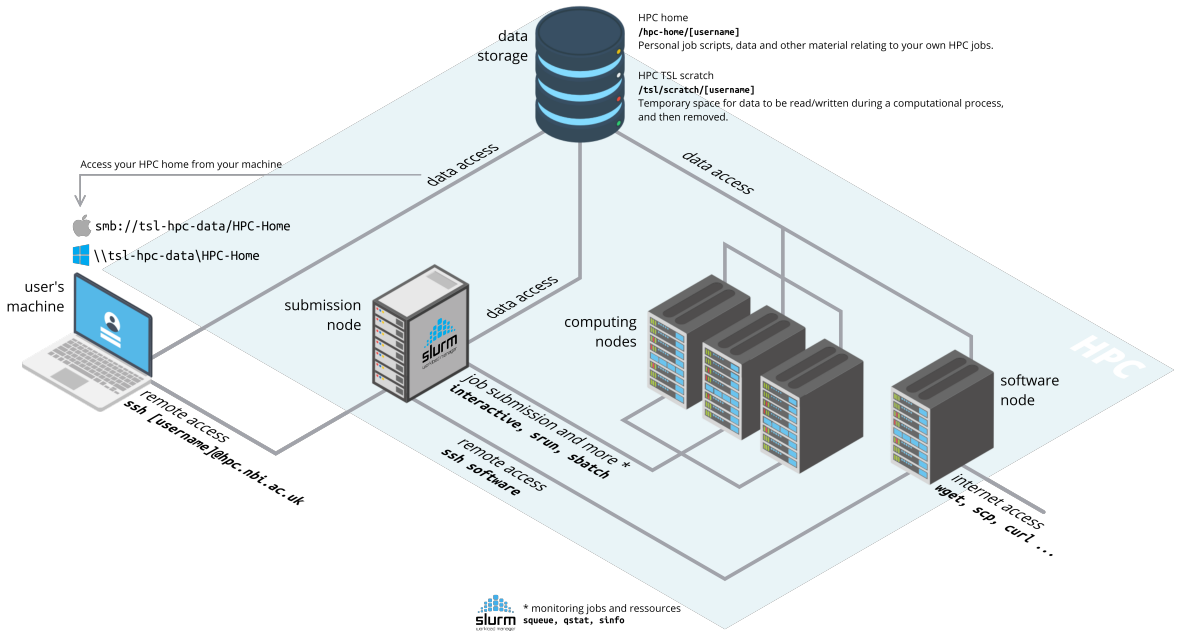


(a) iTerm

1.3 How to access the remote server

To save time, we are going to be working on a remote server where all the necessary data and software available. When we say a 'remote sever', we are talking about a computer that is not the one you are working on right now. You will access the remote server where everything is

prepared for the lesson. We will learn the basics of the shell by manipulating some data files. Some of these files are very large , and would take time to download to your computer. We will also be using several bioinformatic packages in later lessons and installing all of the software would take up time even more time. A 'ready-to-go' sever let's us focus on learning.

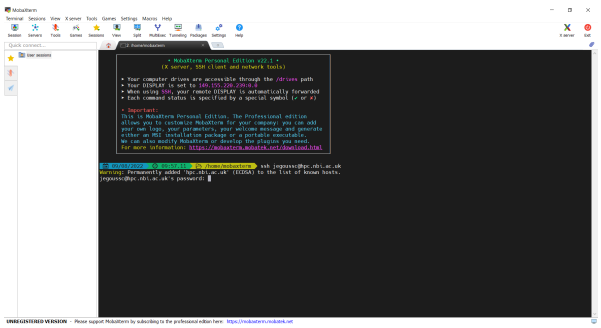


The remote server is a computer cluster used for high performance computing (HPC) for the Norwich BioScience Institutes (NBI). You can log into the remote server using the command `ssh`, your username and the address of the server `hpc.nbi.ac.uk` (make sure to replace `[username]` by your actual TSL username).

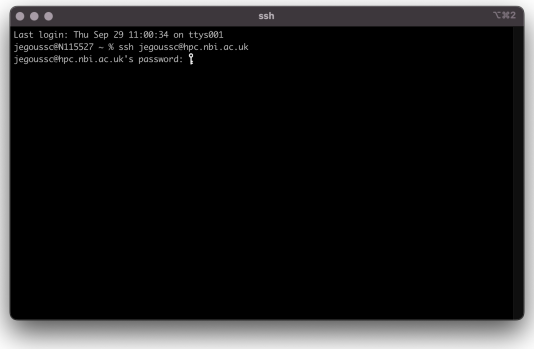
```
ssh [username]@hpc.nbi.ac.uk
```

You must then input your password.

> **Save your password for later**
>
> MobaXterm will offer for you to save your password. It is up to you to do so.



(a) Windows



(a) MacOS

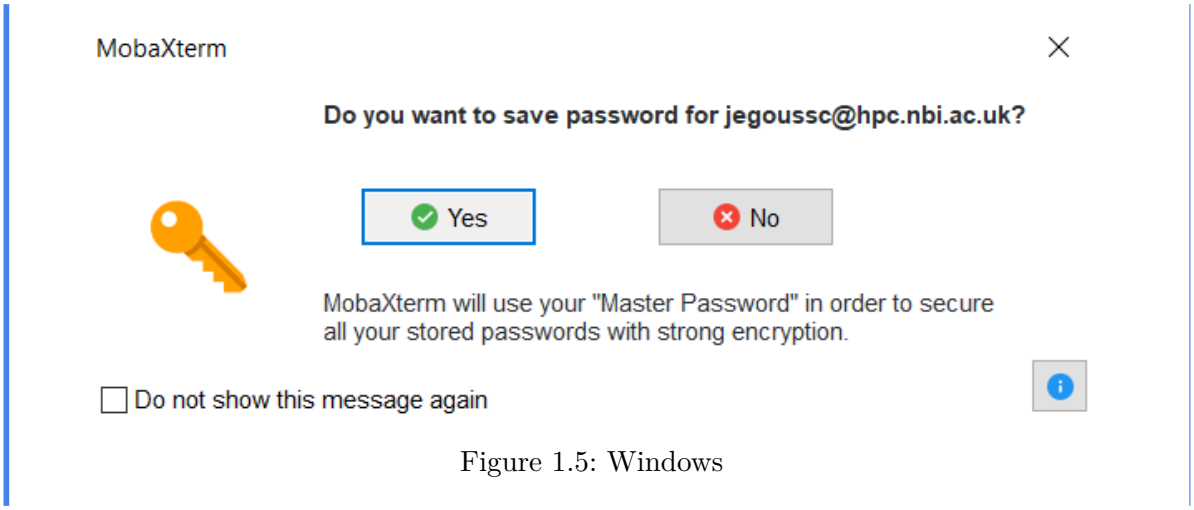


Figure 1.5: Windows

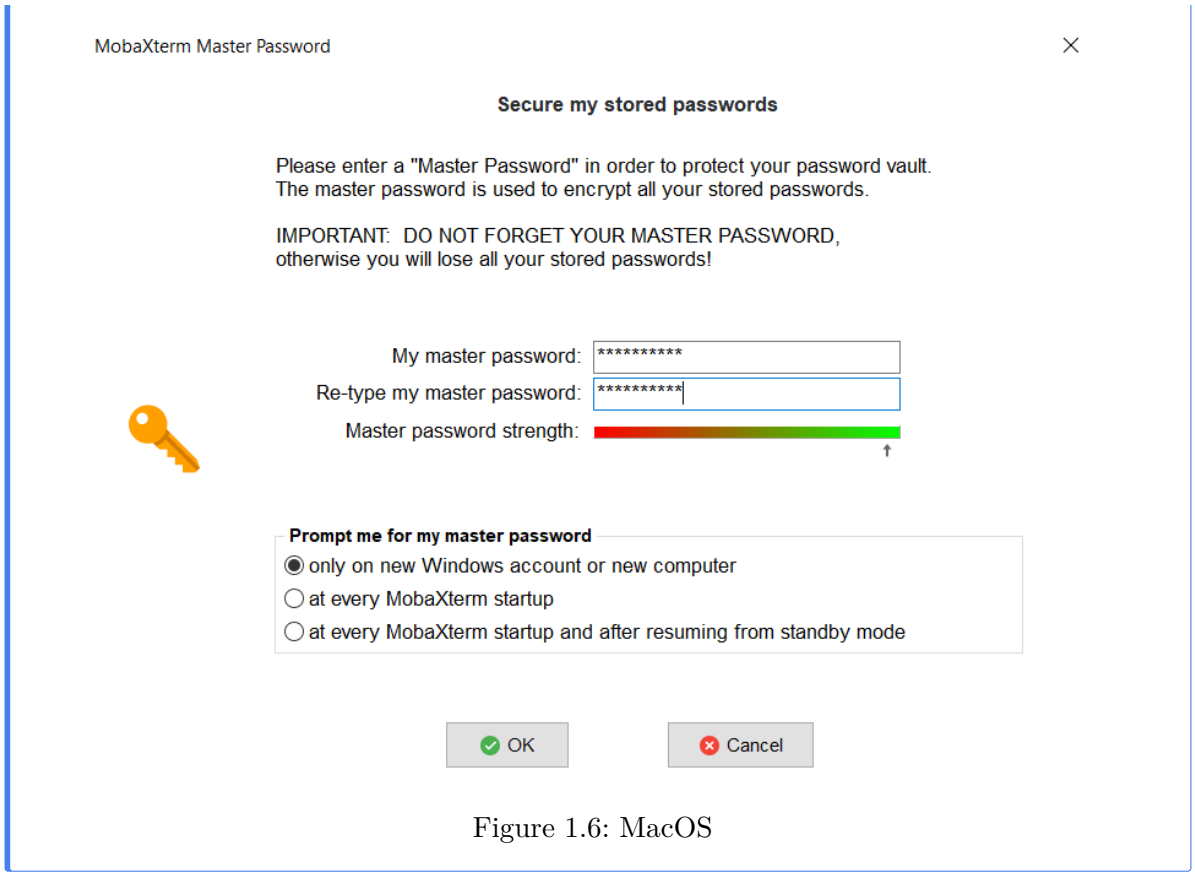
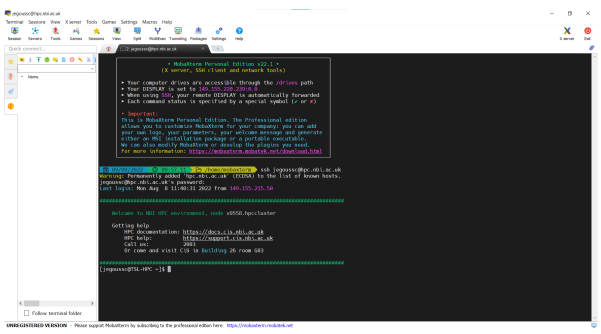
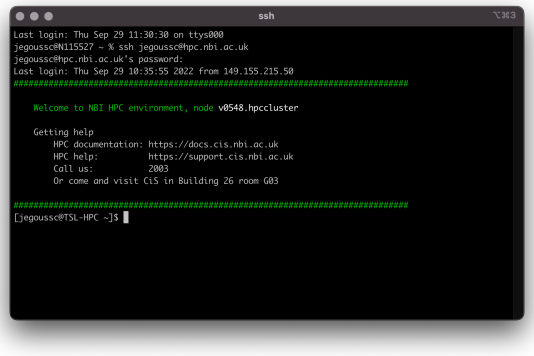


Figure 1.6: MacOS

After entering you password, you will be logged in the server and see the welcome message as seen in the screen capture below:



(a) Windows



(a) MacOS

This welcome message provides a lot of information about the remote server that you're logging into. We're not going to use most of this information for our workshop, so you can clear your

screen using the `clear` command.

Type the word `clear` into the terminal and press the `Enter` key.

This will scroll your screen down to give you a fresh screen and will make it easier to read. You haven't lost any of the information on your screen. If you scroll up, you can see everything that has been output to your screen up until this point.

> Tip
>
> Hot-key combinations are shortcuts for performing common commands. The hot-key combination for clearing the console is `Ctrl+L`. Feel free to try it and see for yourself.

1.4 Navigating your file system

The part of the operating system that manages files and directories is called the **file system**. It organizes our data into files, which hold information, and directories (also called "folders"), which hold files or other directories.

Several commands are frequently used to create, inspect, rename, and delete files and directories.

1.4.1 Prompt

The dollar sign is a **prompt**, which shows us that the shell is waiting for input (other shell may use a different character as a prompt and may add information before the prompt). When typing commands, either from these lessons or from other sources, do not type the prompt, only the commands that follow it.

```
$
```

1.4.2 Print working directory

Let's find out where we are by running a command called `pwd` (which stands for "**p**rint **w**orking **d**irectory"). At any moment, our **current working directory** is our current default directory, i.e., the directory that the computer assumes we want to run commands in, unless we explicitly specify something else. Here, the computer's response is `/hpc-home/[username]`, which is the top level directory within our server:

```
$ pwd
/hpc-home/[username]
```

1.4.3 Listing

Let's look at how our file system is organized. We can see what files and subdirectories are in this directory by running `ls`, which stands for "listing":

```
$ ls
```

`ls` prints the names of the files and directories in the current directory in alphabetical order, arranged neatly into columns. We'll be working within the `shell_data` subdirectory, and creating new subdirectories, throughout this workshop.

1.4.4 Make new directory

Let's make a new directory (folder) called `shell_data` for today's session using the command `mkdir` for "**m**a**k**e **dir**ectory":

```
$ mkdir shell_data
```

Now let's see our new directory using the listing command `ls`:

```
$ ls
```

1.4.5 Copy directory

Our home directory is empty. Let's copy a directory called `shell_data` from the home directory of user `jegoussc` using the command `cp` for "**c**o**p**y". The flag `-r` stands for "recursive" and specifies that we want to copy the full content of the directory `shell_data`:

```
$ cp -r /tsl/data/shell_data/ /hpc-home/[username]/
```

1.4.6 Change directory

The command to change locations in our file system is `cd`, followed by a directory name to change our working directory. `cd` stands for "**c**hange **d**irectory".

```
$ cd shell_data
```

We can also use the command `pwd` to print the current working directory:

```
$ pwd
```

1.4.7 Flags

Earlier, we used the flag `-r` with the `cp` command. Let's have a look at the content of the directory now. We can make the `ls` output more comprehensible by using the **flag** `-F`, which tells `ls` to add a trailing `/` to the names of directories:

```
$ ls -F
sra_metadata/  untrimmed_fastq/
```

Anything with a "/" after it is a directory. Things with a "*" after them are programs. If there are no decorations, it's a file.

1.4.8 Manuals

`ls` has lots of other options. To find out what they are, we can type:

```
$ man ls
```

`man` (short for "**man**ual") displays detailed documentation (also referred as man page or man file) for `bash` commands. It is a powerful resource to explore `bash` commands, understand their usage and flags. Some manual files are very long. You can scroll through the file using your keyboard's down arrow or use the `Space` key to go forward one page and the b key to go backwards one page. When you are done reading, hit the key `q` to quit.

> Exercise
>
> Use the `-l` option for the `ls` command to display more information for each item in the directory. What is one piece of additional information this long format gives you that you don't see with the bare `ls` command?

> Solution
>
> ```
> $ ls -l
> ```

No one can possibly learn all of these arguments, that's what the manual page is for. You can (and should) refer to the manual page or other help files as needed.

Let's go into the `untrimmed_fastq` directory and see what is in there.

```
$ cd untrimmed_fastq
$ ls -F
SRR097977.fastq*  SRR098026.fastq*
```

This directory contains two files with `.fastq` extensions. FASTQ is a format for storing information about sequencing reads and their quality. We will be learning more about FASTQ files in a later lesson.

1.4.9 Tab Completion

Typing out file or directory names can waste a lot of time and it's easy to make typing mistakes. Instead we can use tab complete as a shortcut. When you start typing out the name of a directory or file, then hit the Tab key, the shell will try to fill in the rest of the directory or file name.

Return to your home directory:

```
$ cd
```

then enter (`<tab>` to hit the tabulation key):

```
$ cd she<tab>
```

The shell will fill in the rest of the directory name for `shell_data` and auto-complete the command:

```
$ cd shell_data
```

Now change directories to `untrimmed_fastq` in `shell_data`

```
$ cd untrimmed_fastq
```

Using tab complete can be very helpful. However, it will only auto-complete a file or directory name if you've typed enough characters to provide a unique identifier for the file or directory you are trying to access.

For example, if we now try to list the files which names start with `SR` by using tab complete:

```
$ ls SR<tab>
```

The shell auto-completes your command to `SRR09`, because all file names in the directory begin with this prefix. When you hit Tab again, the shell will list the possible choices.

```
$ ls SRR09<tab><tab>
SRR097977.fastq  SRR098026.fastq
```

Tab completion can also fill in the names of programs, which can be useful if you remember the beginning of a program name.

```
$ pw<tab><tab>
pwck      pwconv    pwd       pwdx      pwunconv  pwmake  pwscore
```

Displays the name of every program that starts with `pw`.

1.5 Summary

We now know how to move around our file system using the command line. This gives us an advantage over interacting with the file system through a GUI as it allows us to work on a remote server, carry out the same set of operations on a large number of files quickly, and opens up many opportunities for using bioinformatic software that is only available in command line versions.

In the next few episodes, we'll be expanding on these skills and seeing how using the command line shell enables us to make our workflow more efficient and reproducible.

! Key points

- The shell gives you the ability to work more efficiently by using keyboard commands rather than a GUI.
- Useful commands for navigating your file system include: `ls`, `pwd`, and `cd`.
- Most commands take options (flags) which begin with a `-`.
- Tab completion can reduce errors from mistyping and make work more efficient in the shell.

2 Navigating Files and Directories

Time

- Teaching: 30 min
- Exercises: 20 min

Objectives

- Use a single command to navigate multiple steps in your directory structure, including moving backwards (one level up).
- Perform operations on files in directories outside your working directory.
- Work with hidden directories and hidden files.
- Inter-convert between absolute and relative paths.
- Employ navigational shortcuts to move around your file system.

2.1 Moving around the file system

We've learned how to use `pwd` to find our current location within our file system. We've also learned how to use `cd` to change locations and `ls` to list the contents of a directory. Now we're going to learn some additional commands for moving around within our file system.

Use the commands we've learned so far to navigate to the `shell_data/untrimmed_fastq` directory, if you're not already there.

```
$ cd
$ cd shell_data
$ cd untrimmed_fastq
```

What if we want to move back up and out of this directory and to our top level directory? Can we type `cd shell_data`? Try it and see what happens.

```
$ cd shell_data
-bash: cd: shell_data: No such file or director
```

Your computer looked for a directory or file called `shell_data` within the directory you were already in. It didn't know you wanted to look at a directory level above the one you were located in.

We have a special command to tell the computer to move us back or up one directory level.

```
$ cd ..
```

Now we can use `pwd` to make sure that we are in the directory we intended to navigate to, and `ls` to check that the contents of the directory are correct.

```
$ pwd
/hpc-home/[username]/shell_data
```

```
$ ls
sra_metadata  untrimmed_fastq
```

From this output, we can see that `..` did indeed take us back one level in our file system.

You can chain these together like so:

```
$ ls ../../
```

prints the contents of `/hpc-home`.

2.2 Hidden directories

Exercise

Find hidden directories.
First navigate to the `shell_data` directory. There is a hidden directory within this directory. Explore the options for `ls` to find out how to see hidden directories. List the contents of the directory and identify the name of the text file in that directory.
Hint: hidden files and folders in Unix start with `.`, for example `.my_hidden_directory`

Solution

First use the `man` command to look at the options for `ls`.

```
$ man ls
```

The `-a` option is short for `all` and says that it causes `ls` to "not ignore entries starting

with ." This is the option we want.

```
$ ls -a
.  ..  .hidden  sra_metadata  untrimmed_fastq
```

The name of the hidden directory is `.hidden`. We can navigate to that directory using `cd`.

```
cd .hidden
```

And then list the contents of the directory using `ls`.

```
$ ls
youfoundit.txt
```

The name of the text file is `youfoundit.txt`.

In most commands the flags can be combined together in no particular order to obtain the desired results/output.

```
$ ls -Fa
$ ls -laF
```

2.3 Examining the contents of other directories

By default, the `ls` commands lists the contents of the working directory (i.e. the directory you are in). You can always find the directory you are in using the `pwd` command. However, you can also give `ls` the names of other directories to view. Navigate to your home directory if you are not already there.

```
$ cd
```

Then enter the command:

```
$ ls shell_data
sra_metadata  untrimmed_fastq
```

This will list the contents of the `shell_data` directory without you needing to navigate there. The `cd` command works in a similar way.

Try entering:

```
$ cd
$ cd shell_data/untrimmed_fastq
```

This will take you to the `untrimmed_fastq` directory without having to go through the intermediate directory.

> Exercise
>
> Navigate to your home directory. From there, list the contents of the `untrimmed_fastq` directory.

> Solution
>
> ```
> $ cd
> $ ls shell_data/untrimmed_fastq/
> SRR097977.fastq  SRR098026.fastq
> ```

2.4 Full vs. Relative Paths

The `cd` command takes an argument which is a directory name. Directories can be specified using either a *relative* path or a full *absolute* path. The directories on the computer are arranged into a hierarchy. The full path tells you where a directory is in that hierarchy. Navigate to the home directory, then enter the `pwd` command.

```
$ cd
$ pwd
/hpc-home/[username]
```

This is the full name of your home directory. This tells you that you are in a directory called `[username]`, which sits inside a directory called `hpc-home` which sits inside the very top directory in the hierarchy. The very top of the hierarchy is a directory called `/` which is usually referred to as the *root directory*. So, to summarize: `[username]` is a directory in `hpc-home` which is a directory in `/`. More on `root` and `home` in the next section.

Now enter the following command:

```
$ cd /home/[username]/shell_data/.hidden
```

This jumps forward multiple levels to the `.hidden` directory. Now go back to the home directory.

```
$ cd
```

You can also navigate to the `.hidden` directory using:

```
$ cd shell_data/.hidden
```

These two commands have the same effect, they both take us to the `.hidden` directory. The first uses the absolute path, giving the full address from the home directory. The second uses a relative path, giving only the address from the working directory. A full path always starts with a `/`. A relative path does not.

A relative path is like getting directions from someone on the street. They tell you to "go right at the stop sign, and then turn left on Main Street". That works great if you're standing there together, but not so well if you're trying to tell someone how to get there from another country. A full path is like GPS coordinates. It tells you exactly where something is no matter where you are right now.

You can usually use either a full path or a relative path depending on what is most convenient. If we are in the home directory, it is more convenient to enter the full path. If we are in the working directory, it is more convenient to enter the relative path since it involves less typing.

Over time, it will become easier for you to keep a mental note of the structure of the directories that you are using and how to quickly navigate among them.

2.4.1 Navigational Shortcuts

The root directory is the highest level directory in your file system and contains files that are important for your computer to perform its daily work. While you will be using the root (`/`) at the beginning of your absolute paths, it is important that you avoid working with data in these higher-level directories, as your commands can permanently alter files that the operating system needs to function. In many cases, trying to run commands in `root` directories will require special permissions which are not discussed here, so it's best to avoid them and work within your home directory. Dealing with the `home` directory is very common. The tilde character, `~`, is a shortcut for your home directory. In our case, the `root` directory is **two** levels above our `home` directory, so `cd` or `cd ~` will take you to `/home/dcuser` and `cd /` will take you to `/`. Navigate to the `shell_data` directory:

```
$ cd
$ cd shell_data
```

Then enter the command:

```
$ ls ~
```

This prints the contents of your home directory, without you needing to type the full path.

The commands `cd`, and `cd ~` are very useful for quickly navigating back to your home directory. We will be using the `~` character in later lessons to specify our home directory.

2.5 Summary

! Key points

- The `/`, `~`, and `..` characters represent important navigational shortcuts.
- Hidden files and directories start with `.` and can be viewed using `ls -a`.
- Relative paths specify a location starting from the current location, while absolute paths specify a location from the root of the file system.

3 Working with Files and Directories

Time

- Teaching: 30 min
- Exercises: 15 min

Questions

- How can I view and search file contents?
- How can I create, copy and delete files and directories?
- How can I control who has permission to modify a file?
- How can I repeat recently used commands?

Objectives

- View, search within, copy, move, and rename files. Create new directories.
- Use wildcards (`*`) to perform operations on multiple files.
- Make a file read only.
- Use the `history` command to view and repeat recently used commands.

3.1 Working with Files

3.1.1 Our data set of FASTQ files

Now that we know how to navigate around our directory structure, let's start working with our sequencing files. We did a sequencing experiment and have two results files, which are stored in our `untrimmed_fastq` directory.

3.1.2 Wildcards

Navigate to your `untrimmed_fastq` directory:

```
$ cd ~/shell_data/untrimmed_fastq
```

We are interested in looking at the FASTQ files in this directory. We can list all files with the .fastq extension using the command:

```
$ ls *.fastq
SRR097977.fastq  SRR098026.fastq
```

The `*` character is a special type of character called a wildcard, which can be used to represent any number of any type of character. Thus, `*.fastq` matches every file that ends with `.fastq`.

This command:

```
$ ls *977.fastq
SRR097977.fastq
```

lists only the file that ends with `977.fastq`.

This command:

```
$ ls /usr/bin/*.sh
/usr/bin/gettext.sh   /usr/bin/lprsetup.sh          /usr/bin/unix-lpr.sh
/usr/bin/lesspipe.sh  /usr/bin/setup-nsssysinit.sh
```

Lists every file in `/usr/bin` that ends in the characters `.sh`. Note that the output displays **full** paths to files, since each result starts with `/`.

> Exercise
>
> Do each of the following tasks from your current directory using a single `ls` command for each:
>
> 1. List all of the files in `/usr/bin` that start with the letter 'c'.
> 2. List all of the files in `/usr/bin` that contain the letter 'a'.
> 3. List all of the files in `/usr/bin` that end with the letter 'o'.
>
> Bonus: List all of the files in `/usr/bin` that contain the letter 'a' or the letter 'c'.
> Hint: The bonus question requires a Unix wildcard that we haven't talked about yet. Try searching the internet for information about Unix wildcards to find what you need to solve the bonus problem.

Solution

```
$ ls /usr/bin/c*
$ ls /usr/bin/*a*
$ ls /usr/bin/*o

# Bonus
$ ls /usr/bin/*[ac]*
```

3.2 Command History

If you want to repeat a command that you've run recently, you can access previous commands using the up arrow on your keyboard to go back to the most recent command. Likewise, the down arrow takes you forward in the command history.

A few more useful shortcuts:

- `Ctrl` + `C` will cancel the command you are writing, and give you a fresh prompt.
- `Ctrl` + `R` will do a reverse-search through your command history. This is very useful.
- `Ctrl` + `L` or the `clear` command will clear your screen.

You can also review your recent commands with the `history` command, by entering:

```
$ history
```

to see a numbered list of recent commands. You can reuse one of these commands directly by referring to the number of that command.

For example, if your history looked like this:

```
259  ls *
260  ls /usr/bin/*.sh
261  ls *R1*fastq
```

then you could repeat command #260 by entering:

```
$ !260
```

Type `!` (exclamation point) and then the number of the command from your history. You will be glad you learned this when you need to re-run very complicated commands. For more information on advanced usage of `history`, read section 9.3 of Bash manual.

Exercise

Find the line number in your history for the command that listed all the .sh files in `/usr/bin`. Rerun that command.

Solution

First type `history`. Then use `!` followed by the line number to rerun that command.

3.3 Examining Files

We now know how to switch directories, run programs, and look at the contents of directories, but how do we look at the contents of files?

One way to examine a file is to print out all of the contents using the program `cat`, for "con**cat**enate".

Enter the following command from within the `untrimmed_fastq` directory:

```
$ cat SRR098026.fastq
```

This will print out all of the contents of the `SRR098026.fastq` to the screen.

Exercise

1. Print out the contents of the `~/shell_data/untrimmed_fastq/SRR097977.fastq` file. What is the last line of the file?
2. From your home directory, and without changing directories, use one short command to print the contents of all of the files in the `~/shell_data/untrimmed_fastq` directory.

Solution

1. The last line of the file is `C:CCC::CCCCCCCC<8?6A:C28C<608'&&&,'$`.
2. `cat ~/shell_data/untrimmed_fastq/*`

`cat` is a terrific program, but when the file is really big, it can be annoying to use. The program, `less`, is useful for this case. `less` opens the file as read only, and lets you navigate through it. The navigation commands are identical to the `man` program.

Enter the following command:

```
$ less SRR097977.fastq
```

Some navigation commands in `less`:

key	action
Space	to go forward
b	to go backward
g	to go to the beginning
G	to go to the end
q	to quit

`less` also gives you a way of searching through files. Use the "/" key to begin a search. Enter the word you would like to search for and press `enter`. The screen will jump to the next location where that word is found.

Shortcut

Shortcut: If you hit "/" then "enter", `less` will repeat the previous search. `less` searches from the current location and works its way forward. Scroll up a couple lines on your terminal to verify you are at the beginning of the file. Note, if you are at the end of the file and search for the sequence "CAA", `less` will not find it. You either need to go to the beginning of the file (by typing `g`) and search again using `/` or you can use `?` to search backwards in the same way you used `/` previously.
For instance, let's search forward for the sequence `TTTTT` in our file. You can see that we go right to that sequence, what it looks like, and where it is in the file. If you continue to type `/` and hit return, you will move forward to the next instance of this sequence motif. If you instead type `?` and hit return, you will search backwards and move up the file to previous examples of this motif.

Exercise

What are the next three nucleotides (characters) after the first instance of the sequence quoted above?

Solution

`CAC`

Remember, the `man` program actually uses `less` internally and therefore uses the same commands, so you can search documentation using "/" as well!

There's another way that we can look at files, and in this case, just look at part of them. This can be particularly useful if we just want to see the beginning or end of the file, or see how it's formatted.

The commands are `head` and `tail` and they let you look at the beginning and end of a file, respectively.

```
$ head SRR098026.fastq
@SRR098026.1 HWUSI-EAS1599_1:2:1:0:968 length=35
NNNNNNNNNNNNNNNNCNNNNNNNNNNNNNNNNNN
+SRR098026.1 HWUSI-EAS1599_1:2:1:0:968 length=35
!!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!!!
@SRR098026.2 HWUSI-EAS1599_1:2:1:0:312 length=35
NNNNNNNNNNNNNNNNANNNNNNNNNNNNNNNNNN
+SRR098026.2 HWUSI-EAS1599_1:2:1:0:312 length=35
!!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!!!
@SRR098026.3 HWUSI-EAS1599_1:2:1:0:570 length=35
NNNNNNNNNNNNNNNNANNNNNNNNNNNNNNNNNN
```

```
$ tail SRR098026.fastq
+SRR098026.247 HWUSI-EAS1599_1:2:1:2:1311 length=35
#!##!##################!!!!!!######
@SRR098026.248 HWUSI-EAS1599_1:2:1:2:118 length=35
GNTGNGGTCATCATACGCGCCCNNNNNNNGGCATG
+SRR098026.248 HWUSI-EAS1599_1:2:1:2:118 length=35
B!;?!A=5922:##########!!!!!!######
@SRR098026.249 HWUSI-EAS1599_1:2:1:2:1057 length=35
CNCTNTATGCGTACGGCAGTGANNNNNNNGGAGAT
+SRR098026.249 HWUSI-EAS1599_1:2:1:2:1057 length=35
A!@B!BBB@ABAB#########!!!!!!######
```

The `-n` option to either of these commands can be used to print the first or last `n` lines of a file.

```
$ head -n 1 SRR098026.fastq
@SRR098026.1 HWUSI-EAS1599_1:2:1:0:968 length=35
```

```
$ tail -n 1 SRR098026.fastq
A!@B!BBB@ABAB#########!!!!!!######
```

3.4 Details on the FASTQ format

Although it looks complicated (and it is), it's easy to understand the fastq format with a little decoding. Some rules about the format include…

Line	Description
1	Always begins with '@' and then information about the read
2	The actual DNA sequence
3	Always begins with a '+' and sometimes the same info in line 1
4	Has a string of characters which represent the quality scores; must have same number of characters as line 2

We can view the first complete read in one of the files in our dataset by using `head` to look at the first four lines.

```
$ head -n 4 SRR098026.fastq
@SRR098026.1 HWUSI-EAS1599_1:2:1:0:968 length=35
NNNNNNNNNNNNNNNNCNNNNNNNNNNNNNNNNNN
+SRR098026.1 HWUSI-EAS1599_1:2:1:0:968 length=35
!!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!!!
```

All but one of the nucleotides in this read are unknown (`N`). This is a pretty bad read!

Line 4 shows the quality for each nucleotide in the read. Quality is interpreted as the probability of an incorrect base call (e.g. 1 in 10) or, equivalently, the base call accuracy (e.g. 90%). To make it possible to line up each individual nucleotide with its quality score, the numerical score is converted into a code where each individual character represents the numerical quality score for an individual nucleotide. For example, in the line above, the quality score line is:

```
!!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!!!
```

The `#` character and each of the `!` characters represent the encoded quality for an individual nucleotide. The numerical value assigned to each of these characters depends on the sequencing platform that generated the reads. The sequencing machine used to generate our data uses the standard Sanger quality PHRED score encoding, Illumina version 1.8 onwards. Each character is assigned a quality score between 0 and 42 as shown in the chart below.

```
Quality encoding: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJK
                  |         |         |         |         |
Quality score:    0........10........20........30........40..
```

Each quality score represents the probability that the corresponding nucleotide call is incorrect. This quality score is logarithmically based, so a quality score of 10 reflects a base call accuracy of 90%, but a quality score of 20 reflects a base call accuracy of 99%. These probability values are the results from the base calling algorithm and dependent on how much signal was captured for the base incorporation.

Looking back at our read:

```
@SRR098026.1 HWUSI-EAS1599_1:2:1:0:968 length=35
NNNNNNNNNNNNNNNNCNNNNNNNNNNNNNNNNNN
+SRR098026.1 HWUSI-EAS1599_1:2:1:0:968 length=35
!!!!!!!!!!!!!!!!#!!!!!!!!!!!!!!!!!!
```

we can now see that the quality of each of the `N`s is 0 and the quality of the only nucleotide call (`C`) is also very poor (`#` = a quality score of 2). This is indeed a very bad read.

3.5 Manipulating files

Now we can move around in the file structure, look at files, and search files. But what if we want to copy files or move them around or get rid of them? Most of the time, you can do these sorts of file manipulations without the command line, but there will be some cases (like when you're working with a remote computer like we are for this lesson) where it will be impossible. You'll also find that you may be working with hundreds of files and want to do similar manipulations to all of those files. In cases like this, it's much faster to do these operations at the command line.

3.5.1 Copying Files

When working with computational data, it's important to keep a safe copy of that data that can't be accidentally overwritten or deleted. For this lesson, our raw data is our FASTQ files. We don't want to accidentally change the original files, so we'll make a copy of them and change the file permissions so that we can read from, but not write to, the files.

First, let's make a copy of one of our FASTQ files using the `cp` command.

Navigate to the `shell_data/untrimmed_fastq` directory and enter:

```
$ cp SRR098026.fastq SRR098026-copy.fastq
$ ls -F
SRR097977.fastq  SRR098026-copy.fastq  SRR098026.fastq
```

We now have two copies of the `SRR098026.fastq` file, one of them named `SRR098026-copy.fastq`. We'll move this file to a new directory called `backup` where we'll store our backup data files.

3.5.2 Creating Directories

The `mkdir` command is used to make a directory. Enter `mkdir` followed by a space, then the directory name you want to create:

```
$ mkdir backup
```

3.5.3 Moving and renaming

We can now move our backup file to this directory. We can move files around using the command `mv`:

```
$ mv SRR098026-copy.fastq backup
$ ls backup
SRR098026-copy.fastq
```

The `mv` command is also how you rename files. Let's rename this file to make it clear that this is a backup:

```
$ cd backup
$ mv SRR098026-copy.fastq SRR098026-backup.fastq
$ ls
SRR098026-backup.fastq
```

3.5.4 File Permissions

We've now made a backup copy of our file, but just because we have two copies, it doesn't make us safe. We can still accidentally delete or overwrite both copies. To make sure we can't accidentally mess up this backup file, we're going to change the permissions on the file so that we're only allowed to read (i.e. view) the file, not write to it (i.e. make new changes).

View the current permissions on a file using the `-l` (long) flag for the `ls` command:

```
$ ls -l
-rwx------ 1 [username] TSL_20 43332 Aug 11 13:58 SRR098026-backup.fastq
```