Introduction to Command-Line & Scripting For Bioinformatics

Steve Pederson
Stephen Bent
Dan Kortschak

University of Adelaide 24th March, 2016

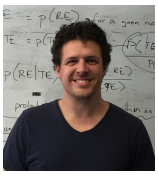
Licensing

Contents

Workshop Information

The Trainers

Mr. Steve Pederson
Co-ordinator
Bioinformatics Hub
The University of Adelaide
South Australia
stephen.pederson@adelaide.edu.au



Dr. Jimmy Breen
Bioinformatician
Bioinformatics Hub & Robinson Research Institute
The University of Adelaide
South Australia
jimmy.breen@adelaide.edu.au

Dr. Hien To
Bioinformatician
Bioinformatics Hub
The University of Adelaide
South Australia
hien.to@adelaide.edu.au

Mr Alastair Ludington
Bioinformatician
Bioinformatics Hub
The University of Adelaide
South Australia
alastair.ludington@adelaide.edu.au

Ms Ramona Rogers
Computing Officer
IT Support
The University of Adelaide
South Australia
ramona.rogers@adelaide.edu.au

Welcome

Thank you for your attendance & welcome to the Introduction to Command Line & Shell Scripting Workshop. This is an offering by the University of Adelaide, Bioinformatics Hub which is a centrally funded initiative from the Department of Vice-Chancellor (Research), with the aim of assisting & enabling researchers in their work. Training workshops & seminars such as this one are an important part of this initiative. The Bioinformatics Hub itself has a web-page at http://www.adelaide.edu.au/bioinformatics-hub/, and **to be kept up to date on upcoming events and workshops, please join the internal Bioinformatics mailing list on** http://list.adelaide.edu.au/mailman/listinfo/bioinfo.

The Bioinformatics Hub has just started a Twitter account, so please follow us on (https://twitter.com/UofABioinfoHub/). We also have an active Slack team for discussing Bioinformatics questions with the local community. Slack teams do require an invitation to join, so please email the Hub on bioinf_hub@adelaide.edu.au to join the community. All are welcome.

Today's workshop has been put together based on previous material and courses prepared by Dr Stephen Bent (*University of Queensland*), with generous technical support & advice provided by Dr Nathan Watson-Haigh (*ACPFG*) and Dr Dan Kortschak (*Adelaide University, Adelson Research Group*). We hope it will be useful in enabling you to continue and to advance your research.

Course Summary

In today's workshop, the morning session will be spent introducing you to the basic tools and concepts required for data handling. In the afternoon session we'll develop these skills to a more advanced level, with progress in both sessions being made at your own pace. Some people may finish early today, but the majority of you probably won't. The VMs you use will be active for the next two weeks should you wish to continue working through the material after the workshop. We will also be using the same VMs for next week's workshop *Introduction to Next Generation Sequencing (NGS) Data*.

The majority of data handling and analysis required in the field of bioinformatics uses the *command line*, alternatively known as the terminal or the *bash shell*. This is a text-based interface in which commands must be typed, as opposed to the Graphical User Interfaces (aka GUIs) that most of us have become accustomed to. Being able to access your computer using these tools enables you to more fully utilise the power & capabilities of your machine, for both Linux & Mac operating systems, and to a lesser extent will even enable you to dig deeper on a Windows system.

Whilst some of the tools we cover today may appear trivial, they are used on a daily basis by those working in the field. These basic tools are essential for writing what are known as *shell scripts*, which we will begin to cover in the afternoon session. These are essentially simple programs that utilise the inbuilt functions of the shell, and are used to automate processes such as de-multiplexing read libraries, or aligning reads to the genome. A knowledge of this simple type of programming and navigation is also essential for accessing the high-performance computing resources such as `phoenix`.

Using the Post-it Notes

For today's session, you will be provided with 3 post-it notes of differing colours. Please use these to signal whether you need help or not by placing them on your monitors. We will interpret these as:

1. **Red** - Help! I can't make something work
2. **Yellow** - I'm working on something, but haven't made it yet
3. **Green** - I've finished the task I was working on

Providing Feedback

While we endeavour to deliver a workshop with quality content and documentation in a venue conducive to an exciting, well run hands-on workshop with a bunch of knowledgeable and likable trainers, we know there are things we could do better.

Whilst we want to know what didn't quite hit the mark for you, what would be most helpful and least depressing, would be for you to provide ways to improve the workshop. i.e. constructive feedback. After all, if we knew something wasn't going to work, we wouldn't have done it or put it into the workshop in the first place!

Clearly, we also want to know what we did well! This gives us that "feel good" factor which will see us through those long days and nights in the lead up to such hands-on workshops!

Document Structure

We have provided you with an electronic copy of the workshop's hands-on tutorial documents. We have done this for two reasons: 1) you will have something to take away with you at the end of the workshop, and 2) you can save time (mis)typing commands on the command line by using copy-and-paste.

We advise you to use Acrobat Reader to view the PDF. This is because it properly supports some features we have implemented to ensure that copy-and-paste of commands works as expected. This includes the appropriate copy-and-paste of special characters like tilde and hyphens as well as skipping line numbers for easy copy-and-past of whole code blocks.

While you could fly through the hands-on sessions doing copy-and-paste, you will learn more if you use the time saved from not having to type all those commands, to understand what each command is doing!

The commands to enter at a terminal look something like this:

```
1 tophat --solexa-quals -g 2 --library-type fr-unstranded -j \
    annotation/Danio_rerio.Zv9.66.spliceSites -o tophat/ZV9_2cells \
    genome/ZV9 data/2cells_1.fastq data/2cells_2.fastq
```

The following styled code is not to be entered at a terminal, it is simply to show you the syntax of the command. You must use your own judgement to substitute in the correct arguments, options, filenames etc

```
tophat [options]* <index_base> <reads_1> <reads_2>
```

The following icons are used in the margin, throughout the documentation to help you navigate around the document more easily:

 Important

 For reference

 Follow these steps

 Questions to answer

 Warning - STOP and read

 Bonus exercise for fast learners

 Advanced exercise for super-fast learners

Computer Setup

We will all be working on our own computers today, and will be accessing Virtual Machines running the Ubuntu operating system on `phoenix`, which is the University of Adelaide's High Performance Computing (HPC) system. The software client `X2GO` which you will have already installed, enables us to access these machines in a familiar Desktop style, even though the majority of our time will be spent within the terminal.

You will have been allocated a VM with an associated IP address. To connect to your VM, **please follow these instructions carefully**. First, we need to create a session with the basic parameters

1. Open X2GO
2. Enter *IntroductionToBash* as the **Session Name**
3. Enter your *IP address* where it say **Host**
4. Enter the word **hub** as the login. **This must be all lower-case**
5. Select **XFCE** from the drop-down menu under **Session Type**
6. Click OK

Now we have created the session, it will appear in your X2Go on the right. To log onto the VM, we simply click on the session, and enter the password **hub**. **Click OK if you receive a message about a security key**. If this process fails, please place the red post-it note on your monitor.

We advise maximising your X2Go window to replicate sitting at the VM as if it is your local machine.

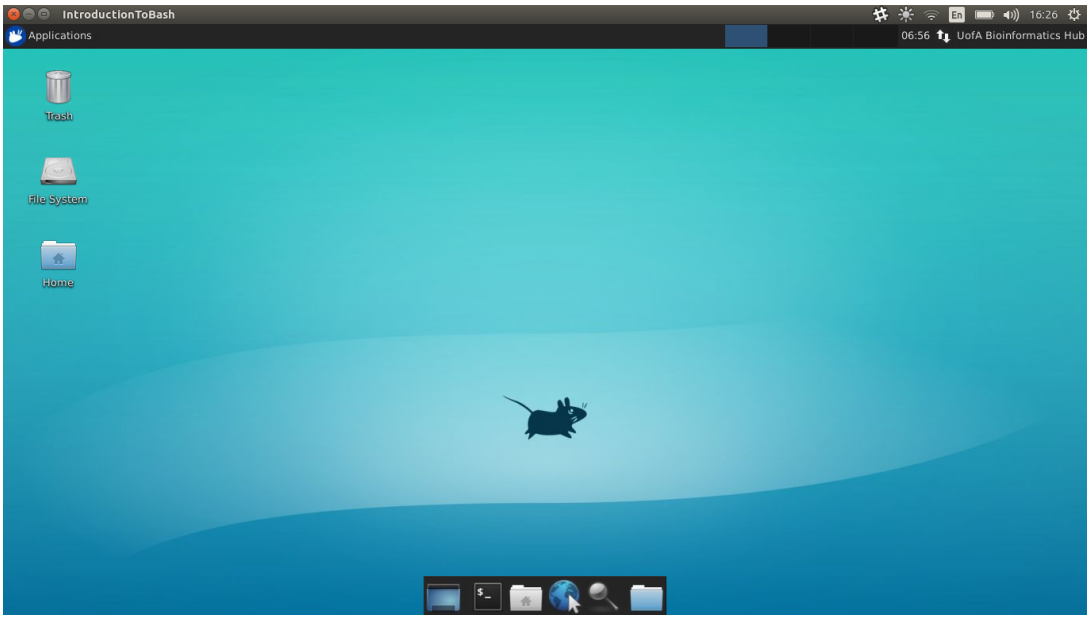


Figure 1: The VM desktop after logon with X2GO

The Ubuntu Desktop

Now that you are connected, you will notice we are in a standard graphical environment. The default Desktop in Ubuntu is Unity, but what we are seeing is one of the many alternatives, known as XFCE. We are using this as it is the most simple for remote connections. As many of us are used to seeing, there are click-able icons on the desktop, and drop-down menus.

Although we won't be using them today, Ubuntu has an built-in Office Suite of programs which you can access from the *Applications* > *Office* menu item. This is where links can be found to open Document Viewer (a .pdf viewer), Libre Office Calc (Excel-like), Libre Office Writer (Word-like) & other standard members of Office Program Suites.

The main interface we will be using today is the `terminal`which can be accessed from the set of icons at the bottom of you screen. **Firefox** can also be accessed from the terminal using the command `firefox &`, by clicking the `Web Browser` icon, or from the drop-down menu in the top left under the group *Internet*.

Introducing The Command Line

Primary Author(s):
Stephen Bent, Robinson Research Institute, University of Adelaide
Steve Pederson, Bioinformatics Hub, University of Adelaide
stephen.pederson@adelaide.edu.au

Contributor(s):
Dan Kortschak, Adelson Research Group, University of Adelaide
dan.kortschak@adelaide.edu.au

Initial Goals

1. Gain familiarity and confidence within the Linux command-line environment
2. Learn how to navigate directories, as well as to copy, move & delete the files within them
3. Look up the name of a command needed to perform a specified task

Background

Command-line tools are the mainstay of analysis of large biological data sets. Good candidate examples for command-line analysis are:

- Manual inspection of fastq, bam & sam files from NGS pipelines
- Automating similar analyses across multiple datasets
- Manipulations of data which are repetitive or laborious to perform manually
- Any analysis that is different from what is available in programs with graphical interfaces (i.e. GUIs).
- Job submission to HPC clusters

Recording all processes as simple scripts also makes an entire analysis more reproducible, as you will have *a record of every procedure you have performed*. It's amazing how often you'll need to revisit something you have performed months ago, and having records of what you've done is immensely useful. From the perspective of an "electronic lab book", this is also very important.

Today we'll explore a few commands to help you gain a little familiarity with some important ones, and to enable you to find help when you're working by yourself. We don't expect you to remember all the commands & options from today. The important thing is to become familiar with the basic syntax for commands, how to put them together, and where to look for help when you're unsure.

Finding your way around

Firstly we need to open a terminal, so **click on the terminal icon** at the bottom of the desktop. There are several variants of the shell, such as the *Bourne-again* shell (i.e. `bash`) and the *C-shell* (please feel free to make all the jokes you can think of). The terminal has a library of commands which are built into it at the time of installing the operating system, and which are part of the Bourne-again Shell, or *bash*. (Historically, it's a replacement for the earlier Bourne Shell, written by Stephen Bourne, so the name is actually a hilarious pun.) We'll explore a few of these commands below, and the words shell or bash will often be used interchangeably with the terminal window. Our apologies to any purists. If you've ever heard of the phrase *shell scripts*, this refers to a series of these commands strung together into a text file which is then able to be executed as a single command.

When you open the terminal on your VM, you'll see the phrase `hub@biohub:~`. This is simply username@computername:, and you may remember you logged on as `hub` during the setup. The tilde (~) represents a shorthand for your home directory, but we'll explore this more later.

Where are we?

When navigating through the folders on any computer, we are all very familiar with clicking through from one folder to another. The folder currently being displayed is usually given in the header of the folder view, so we know where we are currently looking. When using `bash`, we don't click on anything so we need to type commands to change directories, and inspect the contents.

The first thing we need to do is find where we are, when we open `bash`. Type the command `pwd` in the terminal and you will see the output `/home/hub` appear.

```
1 pwd
```

The command `pwd` is what we use for **p**rinting the current (i.e. **w**orking) **d**irectory. Printing in this context means to print some information to the terminal, as opposed to a physical printer. This style of printing harks back to the days when laser printers were not commonplace. Printing information to the terminal itself is what we refer to as printing to the *standard output* or `stdout`.

The directory path that appeared (`/home/hub`) is what will be referred to as your *home directory* for the remainder of the workshop. This is also the information that the tilde (~) represents as a shorthand version, so whenever you see the tilde in a directory path, this is interpreted as meaning `/home/hub`. The *working directory* simply refers to the directory on the computer where you are currently looking, and can be thought of as being the room you are in. We are very familiar with this concept graphically but instead of using a graphical view, we simply have a text-based view.

Looking at the Contents of a Directory

There is another built-in command "`ls`" that we can use to **l**i**s**t the contents of a directory. Enter the `ls` command as it is and it will list the contents of the current directory.

```
1 ls
```

Now open your home folder using the icon at the bottom to give the more familiar, graphical view and compare the contents.

The Linux File System

In the above command, the home directory began with a slash, i.e. `/`. On a Linux-based system, this is considered to be the **root directory** of the file system. Windows users would be more familiar with seeing `C:\` as the root of the file system, and this is a very important difference in the two directory structures. Note also that whilst Windows uses the backslash (\) to indicate a new directory, a Linux-based system uses the forward slash (/), or more commonly just referred to simply as "slash", marking another but very important difference between the two.

Although we haven't directly discovered it yet, a Linux-based file system such as Ubuntu or Mac OS-X is also *case-sensitive*, whilst Windows is not. For example, the command `PWD` is completely different to `pwd` and if `PWD` is the name of a command which has been defined in your shell, you will get completely different results than from the intended `pwd` command.

Spaces are also highly important in Linux, so take note of them where-ever they appear in the given commands.

Changing Directories

The command `pwd` is an example of a *command* that is built into the shell. Another built-in command is `cd` which we use to **c**hange **d**irectory. This changes the directory we are looking in, just like clicking our way through a directory structure in the familiar graphical style we all know well. No matter where we are in a file system, we can move up a directory in the hierarchy by using the command

```
1 cd ..
```

The string "`..`" is the convention for "*one directory above*", whilst a single dot represents the current directory.

Enter the above command and notice that the location immediately to the left of the $ is now given as `/home`. This is also what will be given as the output if we enter the command `pwd`. Note that this given directory is because your personal home folder is within the folder `/home` which contains all of the home folders for all users on the computer. If we now enter `cd ..` one more time we will be in the root directory of the file system. Try this and print the working directory again. As detailed earlier, the output should be the root directory given as `/`.

We can change back to your personal home folder by entering one of either:

```
cd /home/hub
```

or

```
cd ~
```

or even just

```
cd
```

We can also move through multiple directories in one command by separating them with the forward slash "`/`". For example, we could also get to the root directory from our home directory by typing

```
cd ../../
```

Using the above process, return to your home directory `/home/hub`.

Viewing Directories Without Changing Where We Are

Alternatively, we can specify which directory we wish to view the contents of, without having to change into that directory. We simply type the `ls` command, followed by a space, then the directory we wish to view the contents of. To look at the contents of the root directory of the file system (i.e. `/`), we simply add that directory after the command `ls`.

```
ls /
```

Here you can see a whole raft of directories which contain the vital information for the computer's operating system. Among them should be the `/home` directory which is one level above your own home directory, and where the home directories for **all users** are

located, as mentioned earlier.

Try to think of two ways we could inspect the contents of the `/home` directory from your own home directory.

Notice that there are more entries in the `/home` directory. One is your home directory (`/home/hub`), whilst the other is the default home directory that came with the VM (`/home/ubuntu`) before we set up your home directory for today's workshop. Also note that if we ever refer to this higher-level directory, we will write it as `/home`, whereas your personal home directory is always referred to without the preceding forward-slash, and will also be written in normal text font instead of the font we specifically use for code.

Relative vs Absolute Paths

Also note that in your output from `pwd` the resulting path started with the slash, i.e. `/home/hub`. This indicates that it is an **absolute path** as it began with the root of the file system "/". If the slash was missing, it would refer to a sub-directory of the current working directory, and this is what we refer to as a **relative path**. This is an important point which will hopefully become more clear throughout the session.

Another simple example, is that from your home folder (`/home/hub`), the folder `Documents` can be called just using `Documents`. If you are in another folder, you'd probably have to use the full (or absolute) path, which is `/home/hub/Documents`.

A Handy Hint

When working in the terminal, you can scroll through your previous commands by using the up arrow to go backward, and the down arrow to move forward. This can be a big time saver if you've typed a long command with a simple typo, or if you have to do a series of similar commands.

Going Even Deeper

So far, the commands we have used were given either without the use of any subsequent arguments, e.g. `pwd` & `ls`, or with a specific directory as the second argument, e.g. `cd ../` & `ls /home`. Many commands have the additional capacity to specify different options as to how they perform, and these options are often specified between the command name, and the file being operated on. Options are commonly a single letter prefaced with a single dash, or a word prefaced with two dashes. The `ls` command can be given with the option `-l` specified between the command & the directory. This options gives the output in what is known as *long listing* format.

Inspect the contents of your home directory using the long listing format. Please make sure you can tell the difference between the characters `l & 1`.

```
1 ls -l ~
```

The above will give a few lines of output & the first line should be something similar to

```
drwxr-xr-x 2 hub hub 4096 mmm dd hh:mm Desktop
```

where `mmm dd hh:mm` are time and date information.

The important thing to notice is that the word `Desktop` at the end of the line will be coloured **blue**, indicating that it is a directory. The letter '`d`' at the beginning of the initial string of codes `drwxr-xr-x` also indicates this fact. Formally, these letters are known as flags which identify key attributes about each file or directory. We can ignore the fine detail in the rest of these flags until this afternoon (or see the bonus section), but the values `rwx` simply refer to who is able to **r**ead, **w**rite or e**x**ecute the contents of the file or directory. These are very helpful attributes for data security & protection against malicious software.

The entries `hub hub` respectively refer to who is the owner of the directory (or file) & to which group they belong. Again, this information won't be particularly relevant to us today, so we can ignore this until later in our programming careers. In brief, on an Ubuntu system you could have every member of your lab group as an individual user, whilst making every member part of a group. File access permissions can then be applied to any file based on who created it, or whether they are a member of your lab group. Finally, the value `4096` is the size of the directory structure in bytes, whilst the date & time refer to when the directory was created.

The flags we saw at the beginning of the entries above have a clearly defined structure. The first entry shows the file type and for most common files, this entry will be the "-" seen above. The next entries are three triplets which refer to 1) the file's owner, 2) the group they belong to & 3) all users.

In some other entries you'll come across, the name of the file will be in **green**, indicating that it is a file not a directory. There will also be a '`-`' instead of a '`d`' at the beginning of the initial string of flags. The remainder of the information is essentially the same as for the directories we've already seen.

There are many more options that we could specify to give a slightly different output from the `ls` command. Two particularly helpful ones are the options `-h` and `-R`. We could have specified the previous command as

```
ls -l -h ~
```

This will change the file size to *"human-readable"* format, whilst leaving the remainder of the output unchanged. Try it & you will notice that where we initially saw `4096` bytes, the size is now given as `4.0K`. This can be particularly helpful for larger files, as most NGS files are very large indeed and seeing a file size in gigabytes will be much more informative.

The option `-R` tells the `ls` command to look through each directory recursively. If we enter

```
ls -l -R ~
```

the output will be given in two sections. The first is what we have seen previously, but following that will be the contents of the directory `/home/hub/Desktop`. It should become immediately clear that the output from setting this option can get very large & long depending on which directory you start from. It's probably not a good idea to enter `ls -l -R /` as this will print out the entire contents of your file system.

In the case of the `ls` command we can actually specify all the above options together in the command

```
ls -lhR ~
```

This can often save some time, but it is worth noting that not all programmers write their commands in such a way that this convention can be followed. The built-in

shell commands are usually fine with this, but many NGS data processing functions do not accept this convention.

Don't Panic!!!

It's easy for things to go wrong when working in the command-line, but if you've accidentally set something running which you need to exit or if you can't see the command prompt, there are some simple options for stopping a process & getting you back on track. Some options to try are:

`Ctrl-c`	kill the current job. This is usually the first port of call when things go wrong. Also see `man kill` or `man killall` for details on how to kill a process.
`Ctrl-d`	end of input. Sometimes `Ctrl-c` doesn't work but this does.

Exploring Commands In More Detail

Most commands we wish to use have a series of options (sometimes called flags) we are able to set. In reality no-one can remember the full suite of available options, so we need to find out how to get this information. In order to help us find what options are able to be specified, every command built-in to the shell has a manual, or a help page which can take some time to get familiar with. These help pages are displayed using the pager known as `less` which essentially turns the terminal window into a text viewer so *we can display text in the terminal window*, but with no capacity for us to edit the text.

To display the help page for `ls` enter the command

```
1 man ls
```

As beforehand, the space between the two is important & in the first word we are invoking the command `man` which then looks for the *manual* associated with the command `ls`. To navigate through the manual page, we need to know a few shortcuts which are part of the `less` pager..

Although we can navigate through the `less` pager using up & down arrows on our keyboards, some helpful shortcuts are:

<`enter`>	go down one line
<`spacebar`>	go down one page (i.e. a screenful)
b	go up (i.e. **b**ackwards one page)
<	go to the beginning of the document
>	go to the end of the document
q	exit (i.e. **q**uit the page)

Look through the manual page for the `ls` command. How could we give the directory contents in long listing format, sorted by file size?

Many software tools create "*hidden*" files by starting their name with a dot. By default, these files won't be displayed using `ls` How could we make `ls` display these files as well as the main set of visible files.

We can actually find out more about the `less` pager by calling it's own `man` page. Type the command

```
1 man less
```

and the complete page will appear. This can look a little overwhelming, so try pressing `h` which will take you to a summary of the shortcut keys within `less`. There are a lot of them, so try out a few to jump through the file.

A good one to experiment with would be to search for patterns within the displayed text by prefacing the pattern with a slash. Try searching for a common word like *"the"* or *"to"* to see how the function behaves, then try searching for something a bit more useful, like the word *"move"*.

Accessing Manuals or Help Pages

As well as entering the command `man` before the name of a command you need help with, you can often just enter the name of the command with the options `-h` or `--help` specified. Note the convention of a single hyphen which indicates an individual letter will follow, or a double-hyphen which indicates that a word will follow. Unfortunately the methods can vary a little from command to command, so if one method doesn't get you the manual, just try one of the others.

Sometimes it can take a little bit of looking to find something and it's important to be realise we won't break the computer or accidentally launch a nuclear bomb when we look around. It's very much like picking up a piece of paper to see what's under it. If you don't find something at first, just keep looking and you'll find it eventually.

Try accessing the manual for the command `man` all three ways. Was there a difference in the output depending on how we asked to view the manual?

Could we access the help page for the command `ls` all three ways?

Some Useful Commands

So far we have explored the commands `pwd`, `cd`, `ls` & `man` as well as the pager `less`. Inspect the `man` pages for the commands in the following table & fill in the appropriate fields. Have a look at the useful options & try to understand what they will do if specified when invoking the command.

Command	Description of function	Useful options
`man`	Display on-line manual	-k (search for keywords if you don't know the command)
`pwd`	Print working directory, i.e show where you are	none commonly used
`ls`	List contents of a directory	-a, -h, -l, -S, -t, -R
`cd`	Change directory	(scroll down in `man builtins` to find `cd`)
`mv`		-b, -f, -u
`cp`		-b, -f, -u
`rm`		-r (careful...)
`rmdir`		
`mkdir`		-p
`cat`		
`wc`		-l
`head`		-n
`tail`		-n
`echo`		-e
`cut`		-d, -f, -s
`sort`		
`uniq`		-c

Tab auto-complete

A very helpful & time-saving tool in the command line is the ability to automatically complete a command, file or directory name using the <`tab` > key. Try typing

```
1 ls /home/h <tab>
```

where <`tab` > represents the tab key.

Notice how the word `hub` is completed automatically! This functionality will automatically fill as far as it can until conflicting options are reached. In this case, there was only one option so it was able to complete all the way to the end of the file path. This enables us to quickly enter long file paths without the risk of typos. Using this trick will save you an

enormous amount of time trying to find why something doesn't work. The most common error we'll see today will be mistakes in file paths caused by people not taking advantage of this trick.

Now enter

```
ls ~/Do <tab>
```

and it will look like the auto-complete is not working. This is because there are two possibilities & it doesn't know which you want. Hit the tab twice and both will appear in the terminal, then choose one. As well as directory paths, you can use this to auto-complete filenames.

This technique can be used to also find command names. Type in `he` followed by two strike of the <`tab` > key and it will show you all of the commands that being with the string `he`, such as `head`, `help` or any others that may be installed on your computer. If we'd hit the <`tab` > key after typing `hea`, then the command `head` would have auto-completed, although clearly this wouldn't have saved you any typing.

Putting It All Together

Now we can use some the above commands to perform something useful.

Creating and Viewing a File

First, let's create a personal directory under `/home/hub` with your first name as the directory name. **Where you see *firstname* below, use your actual firstname.**

```
cd ~
mkdir firstname
```

Now we can change into this directory.

```
cd firstname
```

Create an empty text file. Check the `man` page for `touch` if you're not sure about this line.

```
touch hello.txt
```

We can read it, but it won't have anything in it yet.

```
cat hello.txt
```

Obviously nothing was printed to your terminal in the previous line because the file is

empty. Let's write something to the file, then try reading it again. In the following line, the symbol `>>` places the text **at the end** of whatever is already in the file. As the file is empty, this will just write a single line.

```
1 echo "Hello" >> hello.txt
2 cat hello.txt
```

We can find a whole lot of information about the file.

```
1 wc hello.txt
```

In the previous line, what do the three numbers represent?

When we added the word `Hello` to our file, we used the symbol >> which actually wrote the word to the end of the file. As the file was completely empty, this placed the word in the first line. This is a VERY handy short-cut for writing information to the end of a file

Now let's add more to the file.

```
1 echo "It's me" >> hello.txt
2 cat hello.txt
3 wc hello.txt
```

For most of the above commands we could have used the auto-complete feature of the bash terminal. Did you remember this trick?

Copying and Renaming a File

Later today, we're going to look through a file containing a list of words. It's currently on your VM in the folder `/usr/share/dict` and has the name `cracklib-small`. Let's copy this to your `firstname` folder, in your home directory.

```
1 cp /usr/share/dict/cracklib-small ~/firstname
```

Next we will rename the file. Note, that in bash there is no "rename" tool. Instead we "move" from it's current location to any location we choose, with whatever name we choose. In the following, the location is the same so we are effectively just giving the file a new name.

```
mv ~/firstname/cracklib-small ~/firstname/words
```

We can look at the first 5 lines of the file using

```
head -n5 ~/firstname/words
```

Or we can look at the last 10 lines of the file using

```
tail -n10 ~/firstname/words
```

We could even page through the file using `less`. (Remember to hit `q` to exit the pager.)

```
less ~/firstname/words
```

We can even find how many lines there are in the file by using

```
wc -l ~/firstname/words
```

Did we need to enter the full file path in the above commands, or could we have saved ourselves some effort by changing into the `~/firstname` directory?

Summary

Now we have experience in several key tasks which are some of the most common tasks needed for any bioinformatics analysis, or for any general HPC usage.

1. How to access a manual

2. How to create and navigate through directories

3. How to create and view files

4. How to copy, move and rename files

Space for Personal Notes or Feedback