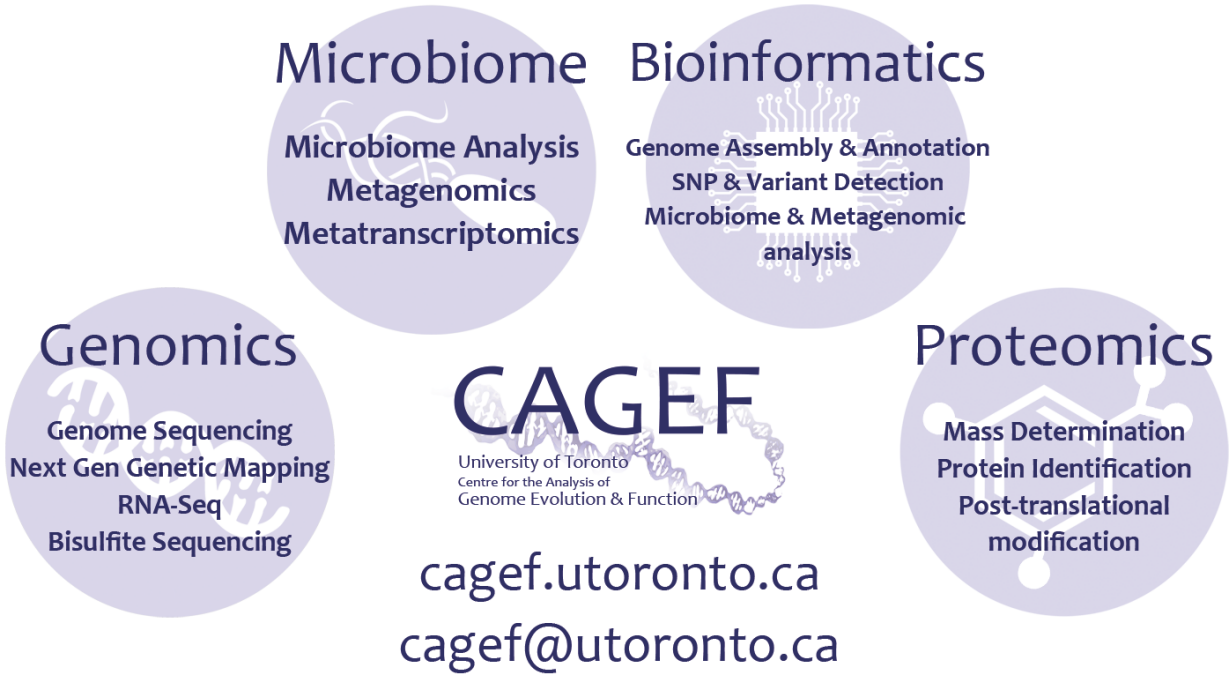


CSB1021HF LEC0131

FUNDAMENTALS OF GENOMIC DATA SCIENCE

0.0.0 Module 4: Introduction to the command line

0.1.0 About Fundamentals of Genomic Data Science

Fundamentals of Genomic Data Science is brought to you by the **Centre for the Analysis of Genome Evolution & Function (CAGEF)** bioinformatics training initiative. This course was developed based on feedback on the needs and interests of the Department of Cell & Systems Biology and the Department of Ecology and Evolutionary Biology.

The structure of this course is a “code-along”, hands-on style! A few hours prior to each lecture, materials will be made available for download at QUERCUS (https://q.utoronto.ca/). The teaching materials will consist of a weekly PDF that you can use to follow along with the instructor along with any datasets that you’ll need to complete the module. This learning approach will allow you to spend the time coding and not taking notes!

As we go along, there will be some in-class challenge questions for you to solve. Post lecture assessments will also be available for each module, building upon the concepts learned in class (see syllabus for grading scheme and percentages of the final mark).

0.1.1 Where is this course going?

We’ll take a blank slate approach here to learning genomic data science and assume you know nothing about programming or working directly with next generation sequencing data. From the beginning of this course to the end we want to guide you from potential scenarios like:

- You don’t know what to do with a set of raw sequencing files fresh from a facility like CAGEF.

- You've been handed a legacy pipeline to analyse your data or maintain for the lab, but you don't know what it runs or how.
- You plan on generating high-throughput data but there are no bioinformaticians around to help you out.

and get you to the point where you can:

- Recognize the basic tools in sequence analysis.
- Plan and write your own data analysis pipelines.
- Explain your data analysis methods to labmates, supervisors, and other colleagues.

0.1.2 How do we get there?

In the first half of this course, we'll focus on how to generate analysis pipelines using the Galaxy platform – a user-friendly graphical interface that provides access to common sequence analysis tools. After we are comfortable with these tools, we'll look at life through the lens of a command-line interface. It is here that we will learn the basics of file manipulation and how to program scripts that can carry out multiple tasks for us. From there we'll revisit tools from the first half and learn skills to make your data analysis life easier.

0.2.0 Goals of the module

1. Learn how to move around your file system at the command line.
2. Learn how to manipulate file contents using basic commands.
3. Learn how to combine a series of commands in an executable bash script.
4. Learn how to regulate access and set permissions for files.
5. Learn how to run operations on a remote server through the command line.

0.3.0 Pre-class modules with Coursera

Each week we strongly encourage you to complete the assigned Coursera modules and/or readings ***before*** class. These are meant to provide you with sufficient background material on each week's module so that we can focus on the act of "doing" something with that data rather than spend a lot of time on the origins of it. You'll find a section outlining the next set of Coursera modules and readings at the end of each module.

0.3.1 Go to www.coursera.org and sign up for an account with your e-mail.

0.3.2 Search the following courses and enroll to audit each course (audit):

- Genomic Data Science with Galaxy, Johns Hopkins University.
- Command Line Tools for Genomic Data Science, Johns Hopkins University.

0.4.0 Setting up your working directory

We suggest that you create a new directory (folder) for this course directly off your root directory called "**FGDS**". Working from your root directory is not necessary, but it will make some of the aspects of the course a little easier to manage. For Mac OS users, we suggest you create this as a subfolder in your **user** directory.

0.4.1 Within this directory, create another directory called "**Module4**". This is where we will store the data used in this week's module.

0.4.2 Create a subdirectory called "**downloads**" to store the initial files as we download them before decompressing and working with them in later steps.

1.0.0 Navigating command-line files

This module is designed as an introduction to some of the most commonly-used commands in the bash shell for genomic data science. Depending on your application, however, there are many shell commands/options that you may find useful. You can find a more detailed summary of some of the most common shell commands in the **section 6.0.0 Appendix**.

Don't be scared to come back to this module to review commands or search new commands online in the future. I've been working at the command line on and off for years and still search and review command options all the time. Most users will not memorize all this stuff! You can also use the `man` command to print the manual for any given command (e.g. `man cd`).

1.0.1 Important command-line wild cards and shortcuts

While working with the command-line there are many tips and tricks we can use to describe or identify files or directories we are searching for. These wildcards are somewhat like regular expressions and let us substitute directly or indirectly for single or multiple filenames.

Command	Description
`*`	Universal wildcard; anything, any number of times (or nothing)
`?`	Universal wildcard; anything, once
`[a-z]`	Any letter or number in range
`{a, b}`	Any letter or number in list
`\`	Escape character; ignore special meaning
`!`	Invert a search

Aside from wildcards, there are also meta-characters or symbols that are used to represent simple ideas or locations when navigating the file system.

Command	Description
`.`	Current directory
`..`	Parent directory (one directory above in hierarchy)
`~`	Root directory (home)

Following these command symbols there are additional key strokes or characters that allow us to alter commands, how they are run, or stop them completely. Along with this list, we have quick keys and commands that can be used to navigate the terminal commands themselves.

Command	Description
`|`	Pipe output from the left-side command into right-side command
`&`	Runs command in the background when placed at the end of the command
`>`	Save the output of a command to a file (e.g. `ls -la > allFilesList.txt`)
`>>`	Append the output of a command to a file (e.g. `ls -la ../ > allFilesList.txt`)
`[Tab]`	Auto complete
`[Tab+Tab]`	List all auto completed files in your path
`[Ctrl+C]`	Cancel a process that is currently running
`[Ctrl+Z]`	Move operation into the background
`clear or [Ctrl+L]`	Clear all commands and results from terminal window
`alias`	Set a short form for a command or series of commands
`[↑/↓]`	Scroll back through the commands you have already entered (aka History)
`[Home] or [End]` `[←/→]`	Scroll quickly to the beginning or end of a command, or character by character. Arrows can be combined with [`Ctrl`] for skipping between space-separated words.

* On Mac OS you may have to substitute the [`Ctrl`] for the [`Cmd`] key

1.0.2 Commands for accessing, creating, and removing content

Now that we've reviewed some basic commands syntax for identifying files and directory structures, we can look closely at the actual manipulation of directories and files.

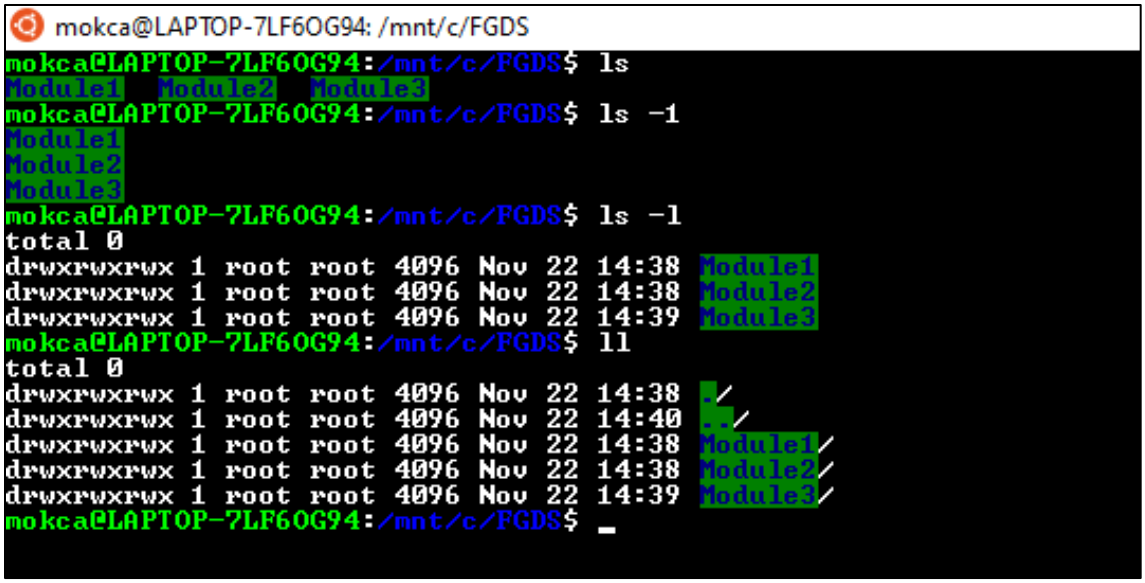
Command/Switches	Description
`pwd`	Print working directory
`ls`	List files in a directory (e.g. ls –al). Run ls --help to access the manual for this command)
`-a`	Include hidden files
`-l`	Show dates and permissions (long listing format).
`-1`	List all files on separate lines
`-S`	List all files in order of descending size
`ll`	An aliased equivalent to the call `ls -alF`
`cd`	Change directory – the primary way of changing your current working directory
`mkdir`	Make new directory
`rmdir`	Remove empty directory
`rm`	Remove file
`-r`	Recursive removal of all files in directory. This option may also be implemented in many other Unix commands.
`-f`	While removing files ignore non-existent files or arguments with prompts (or warnings)
`mv`	Move a file or a folder to a new location or rename a file
`cp`	Copy a file to a new location
`gzip`	Compress files
`gunzip`	Uncompress (extract) files
`tar`	Create or expand a tar archive
`-c`	Create an archive file
`-v`	Verbosely show the progress
`-f`	Filename of archive
`-r`	Append to a specified archive
`-x`	Extract an archive file
`-t`	List files in an archive
`-z`	Filter archive through gzip
`passwd`	Change your password
`history`	Access the commands that you have previously ran
`Ctrl + r`	Search for a previously run command

1.1.0 Explore your command-line directory with `ls`

Let's practice using some of these commands to navigate our directories in our home directory.

1.1.1 Open your command line prompt (Terminal or Bash) and get oriented in your file system using the **`pwd`** and **`ls`** commands.

```
pwd
ls
ls -1           # That's a number
ls -l           # That's a letter
```



1.2.0 Move files and directories with `mv`

The **`mv`** command serves dual purposes, mainly for **mo**v**e**ing files from one place to another. As we'll see, during the process you must also specify a destination folder. You may, however, also define a destination filePath, essentially renaming the file or directory as you move it! The **`mv`** command takes on the form of:

```
mv filePathFrom filePathTo
```

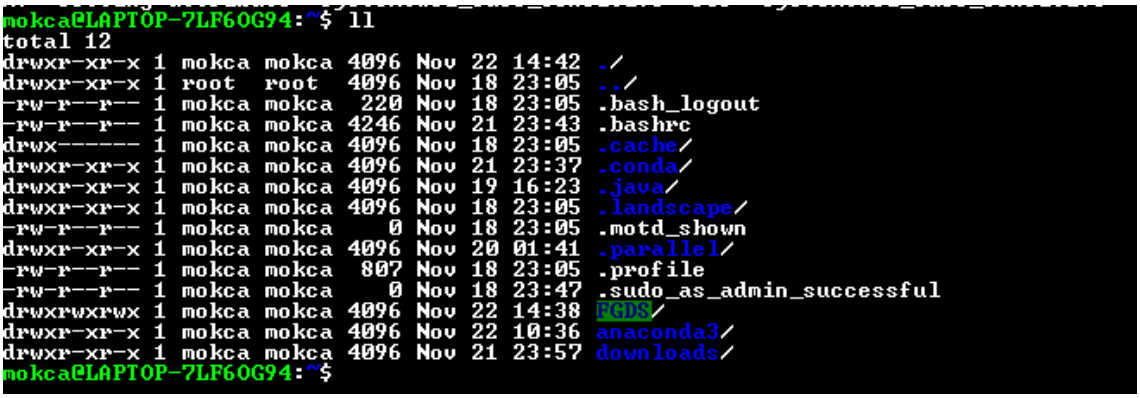
> Note that for directories, if your **`filePathTo`** exists, then the **`filePathFrom`** will be moved into that directory. If **`filePathTo`** does not exist, then the directory will be renamed as it is moved. Additionally, when moving files, destination files can be overwritten or replaced if **`filePathTo`** already exists!

1.2.1 If your "FGDS" directory is not already in your home directory, move it and all its contents there using the **`mv`** command.

If you are working on Windows10, you can access your entire Windows file structure from the "mnt" directory.

1.2.2 Move your FGDS directory if necessary

```
cd ~/                       # Return to your home directory
mv /mnt/c/FGDS ./           # Move over the FGDS folder
ll
```



1.3.0 Change your current directory with `cd`

A mainstay of moving through directories is the **c**hange **d**irectory command – **`cd`**. This will be an important tool in your belt unless you memorize the location of everything in your files.

1.3.1 Use the **`cd`** command to navigate into the **Module1** directory using an absolute path.

```
cd ~/FGDS/Module1
# The location of the FGDS folder will depend on how you've set it up
```

1.3.2 From the **Module1** directory navigate directly to the **Module3** directory using a relative path.

```
cd ../Module3
```

1.3.3 Return to the **home** directory.

```
cd ~/            # 'cd ~' will also work
```

1.4.0 Create new directories with `mkdir`

1.4.1 Create a new directory within **FGDS** for this module called **Module4**. This will require us to use the **`ls`**, **`cd`**, and **`mkdir`** commands, but we will also explore how the **`rmdir`** command works so that you know how to remove a directory. Notice your **Module4** directory is now available in our Finder/File Explorer

```
cd FGDS
ls
mkdir ModuleFour      # Make a directory
ls
rmdir ModuleFour      # Remove the directory (only if empty)
ls
mkdir Module4
ls
ls -la
cd Module4
ls
pwd                   # print the current working directory
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS$ ls
Module1  Module2  Module3
mokca@LAPTOP-7LF6OG94:~/FGDS$ mkdir ModuleFour
mokca@LAPTOP-7LF6OG94:~/FGDS$ ls
Module1  Module2  Module3  ModuleFour
mokca@LAPTOP-7LF6OG94:~/FGDS$ rmdir ModuleFour
mokca@LAPTOP-7LF6OG94:~/FGDS$ ls
Module1  Module2  Module3
mokca@LAPTOP-7LF6OG94:~/FGDS$ mkdir Module4
mokca@LAPTOP-7LF6OG94:~/FGDS$ ll
total 0
drwxrwxrwx 1 mokca mokca 4096 Nov 22 14:50 ./
drwxr-xr-x 1 mokca mokca 4096 Nov 22 14:42 ../
drwxrwxrwx 1 mokca mokca 4096 Nov 22 14:38 Module1/
drwxrwxrwx 1 mokca mokca 4096 Nov 22 14:38 Module2/
drwxrwxrwx 1 mokca mokca 4096 Nov 22 14:39 Module3/
drwxr-xr-x 1 mokca mokca 4096 Nov 22 14:50 Module4/
mokca@LAPTOP-7LF6OG94:~/FGDS$ cd Module4
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ls
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ pwd
/home/mokca/FGDS/Module4
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

1.5.0 Copy files with `cp`

You should now be in the **Module4** directory. Copy **HinfKW20_genomic.fna**, **HinfKW20_cds.fna**, **HinfKW20_rna.fna**, **HinfKW20_protein.faa**, and **HinfKW20_features.txt** files from your **Module1** directory into your **Module4** directory.

In doing so, you will experiment with the **mv**, **cp**, and **rm** commands. We will also explore how to do this all at once with wildcards. Note that both the `.` and `[tab]` meta-characters and commands can make your life a lot easier here by specifying your current location in the filesystem and auto-filling filenames, respectively. You can also scroll up and down previous commands that you want to repeat using the up ↑ and down ↓ arrows.

1.5.1 Copy files individually from **Module1** over to **Module4**

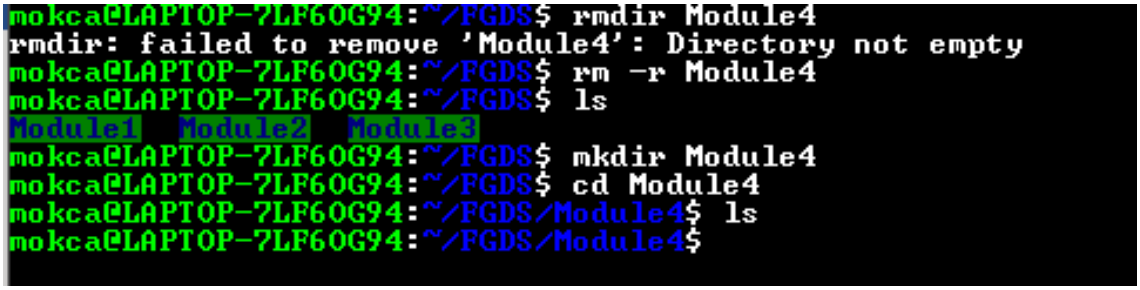
```
pwd              # Make sure you are in the Module4 directory!
ls
ls ../Module1
cp ../Module1/downloads/HinfKW20_genomic.fna .
cp ../Module1/downloads/HinfKW20_cds.fna .
cp ../Module1/downloads/HinfKW20_rna.fna .
cp ../Module1/downloads/HinfKW20_protein.faa .
ls
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ pwd
/home/mokca/FGDS/Module4
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ls
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ls ../Module1
downloads
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ cp ../Module1/downloads/HinfKW20_genomic.fna .
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ cp ../Module1/downloads/HinfKW20_cds.fna .
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ cp ../Module1/downloads/HinfKW20_rna.fna .
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ cp ../Module1/downloads/HinfKW20_protein.faa .
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ls
HinfKW20_cds.fna  HinfKW20_genomic.fna  HinfKW20_protein.faa  HinfKW20_rna.fna
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

1.5.2 Remove the **Module4** directory and remake it

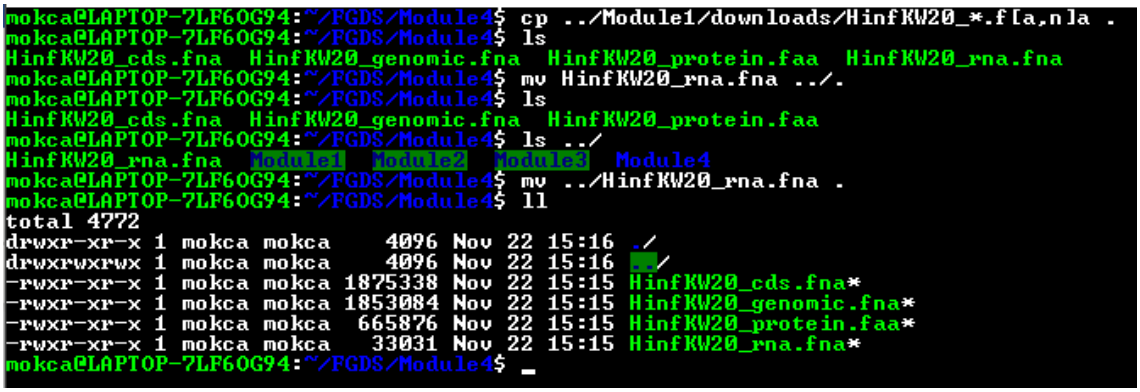
```
cd ..                  # Return to ~/FGDS/
rmdir Module4          # What happens when we use this command?
rm -r Module4
ls
```

```
mkdir Module4
cd Module4
```



1.5.3 Copy all of the .fna and .faa files over from **Module1**.

```
cp ../Module1/downloads/HinfKW20_*.f[a,n]a .
ls
mv HinfKW20_rna.fna ../
ls
ls ../
mv ../HinfKW20_rna.fna .
ll
```



1.6.0 Create a compressed archive with `tar` or `gzip`

You might recall that through working with our data in **Module1-3**, many of our downloads come in compressed and/or archived files. It's time for us to practice using some of the command-line tools to create and work with these space-saving files. The **`tar`** (tape archive) command provides a way to create and extract from archive files but does not *directly* compress them. The tar command takes the form of:

```
tar -switches archiveName archiveContents
```

Some useful **`tar`** command switches:

Switch/flag	Description
`-f --file`	The file or archive you want to specify
`-t --list`	List the files present in an archive
`-c --create`	Create an archive
`-x --extract`	Rather than build an archive, you can use this tag to *extract* an archive.
`-v --verbose`	Display all of the file names in the archive
`-z --gzip`	Filter the archive through gzip (especially great for tar.gz files). Both for compression and decompression depending on other flags!
`-k --keep-old-files`	Don't replace existing files when extracting

1.6.1 Create a **tar** archive that contains all the gene, rna, and protein sequence information that you just moved to your **Module4** directory.

```
ll
tar --help
tar -cvf HinfKW20_sequences.tar HinfKW20_*
# So we want to create an archive with the specified file name
# Note how the f is on the end!
ll
```

```
mokca@LAPTOP-7LF60G94:~/FGDS/Module4$ tar -cvf HinfKW20_sequences.tar HinfKW20_*
HinfKW20_cds.fna
HinfKW20_genomic.fna
HinfKW20_protein.faa
HinfKW20_rna.fna
mokca@LAPTOP-7LF60G94:~/FGDS/Module4$ ll
total 9124
drwxr-xr-x 1 mokca mokca    4096 Nov 22 15:47 ./
drwxrwxrwx 1 mokca mokca    4096 Nov 22 15:16 ../
-rwxr-xr-x 1 mokca mokca 1875338 Nov 22 15:15 HinfKW20_cds.fna*
-rwxr-xr-x 1 mokca mokca 1853084 Nov 22 15:15 HinfKW20_genomic.fna*
-rwxr-xr-x 1 mokca mokca  665876 Nov 22 15:15 HinfKW20_protein.faa*
-rwxr-xr-x 1 mokca mokca   33031 Nov 22 15:15 HinfKW20_rna.fna*
-rw-r--r-- 1 mokca mokca 4433920 Nov 22 15:47 HinfKW20_sequences.tar
```

The `gzip` command has one function – to compress or expand files using the `gzip` algorithm. When we are using `gzip` on multiple files, it will accept them as input but it will individually compress them. By default, it will replace the original file(s) with the gzipped versions (.gz).

1.6.2 Create a compressed `gzip` version of **HinfKW20_cds.fna** and **HinfKW20_genomic.fna**. Take a look at the resulting files. Now decompress the same files with `gunzip`.

```
gzip --help
gzip HinfKW20_cds.fna HinfKW20_genomic.fna
ll
gunzip *.gz
```

1.6.3 Create a compressed `gzip` version of the tar archive. By archiving before compressing the file, everything is kept together as a group!

```
gzip HinfKW20_sequences.tar
ll
```

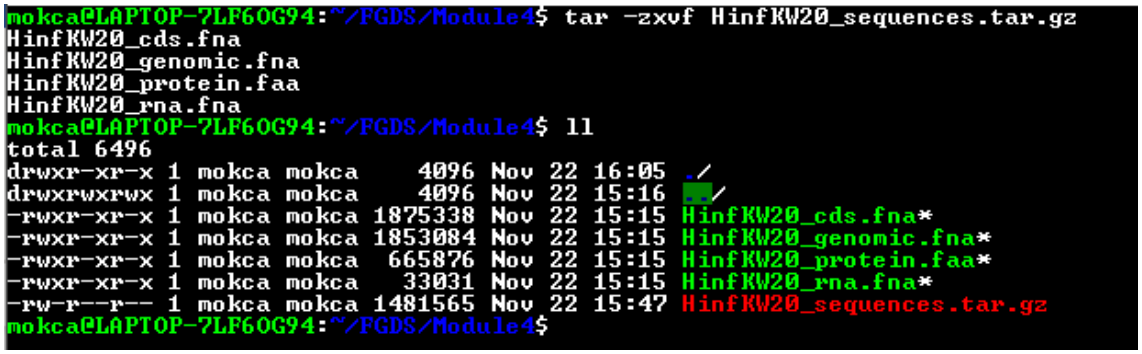
```
mokca@LAPTOP-7LF60G94:~/FGDS/Module4$ gzip HinfKW20_sequences.tar
mokca@LAPTOP-7LF60G94:~/FGDS/Module4$ ll
total 1792
drwxr-xr-x 1 mokca mokca    4096 Nov 22 15:52 ./
drwxrwxrwx 1 mokca mokca    4096 Nov 22 15:16 ../
-rw-r--r-- 1 mokca mokca 1481565 Nov 22 15:47 HinfKW20_sequences.tar.gz
mokca@LAPTOP-7LF60G94:~/FGDS/Module4$
```

1.6.4 Check how much our original archive folder was compressed vs the original version.

```
gzip -l HinfKW20_sequences.tar.gz
ls -l
```

1.6.5 Unarchive and decompress your **tar.gz** file by using `tar` and filtering it through `gzip`.

```
rm HinfKW20_*.f*                          # remove the current files
ll
                                          # x stands for extract!
ls -l
rm HinfKW20_sequences.tar.gz
ll
```



1.6.6 Prior to this we used both **`tar`** and **`gzip`** to accomplish the creation of a tar.gz file but much like our last call, we can create a compressed archive **tar.gz** file by using **`tar`** and filtering it through **`gzip`**.

```
                                          
ls -l
```

1.6.7 How do we view the contents of a tar.gz file? We already know how to do it with a .gz file. Let's compare

```
gzip -l HinfKW20_sequences.tar.gz         # Show the compressed contents
                                          # Show the archive contents
                                          # Show archive content info
rm HinfKW20_sequences.tar.gz
ll
```

2.0.0 Viewing and manipulating file contents

Now that you have a sense of how you can move around your file system to create and remove content, let's look at some important commands for viewing and manipulating file contents. There are different terminal commands that can be used to quickly view, edit, summarize, combine, and search content.

2.1.0 Use the `less` command to view content

The **`less`** command allows you to view the contents of a file by moving through it both forward and backwards. It comes equipped with hot-keys or commands that make it easy to navigate your text files one at a time.

Keys	Description
`[Arrow keys], [space], [page-up], [page-down]`	Navigate around the page in a forwards or backwards direction
`b`	Return to previous page
`Line_numg`	Jump to a line number, default is the start of the file (e.g. 1000g)
`Line_numG`	Jump to a line number, default is the end of the file (e.g. 46G)
`/`	Search for text in the file (e.g. /ribonuc)
`n`	Find next occurrence
`?`	Find previous occurrence
`-N`	Display the line numbers in your file
`q`	Exit the file

2.1.1 Copy the **HinfKW20_features.txt** file from your **Module1** directory into your **Module4** directory with the **`cp`** command. Then create a copy of this file titled **HinfKW20_features_copy.txt**

2.1.2 Explore the **HinfKW20_features_copy.txt** file with the **`less`** command. Jump to line **1000** and then search for the genes **mutS** and **2-isopropylmalate**.

```
less HinfKW20_features_copy.txt
     -N                                  # Display line numbers in the file
     1000g                               # Go to line 1000
     /mutS                               # Look for the mutS gene
     /2-isopropylmalate                  # Look for 2-isopropylmalate
     q                                   # quit less
```

```
2017 CDS     with_protein    GCA_000027305.1 Primary Assembly       chromosome            L42023.1        10433
2017 07 1044902 +       AAC22647.1                    2-isopropylmalate synthase (leuA)                      HI
2017 _0986 1596    531
2018 gene    protein_coding  GCA_000027305.1 Primary Assembly       chromosome            L42023.1        10449
2018 88 1046064 +                                                    HI_0987 1077
2019 CDS     with_protein    GCA_000027305.1 Primary Assembly       chromosome            L42023.1        10449
2019 88 1046064 +       AAC22648.1                    3-isopropylmalate dehydrogenase (beta-IPM dehydrogenase) (
2019 leuB)                HI_0987 1077    358
2020 gene    protein_coding  GCA_000027305.1 Primary Assembly       chromosome            L42023.1        10462
2020 40 1047649 +                                                    HI_0988 1410
2021 CDS     with_protein    GCA_000027305.1 Primary Assembly       chromosome            L42023.1        10462
2021 40 1047649 +       AAC22649.1                    3-isopropylmalate dehydratase, alpha subunit (leuC)
2021             HI_0988 1410    469
2022 gene    protein_coding  GCA_000027305.1 Primary Assembly       chromosome            L42023.1        10476
2022 73 1048275 +                                                    HI_0989 603
```

2.1.3 Comprehension challenge: on what line would you find "galactoside ABC transporter, ATP-binding protein"?

2.2.0 Use `vi` to edit your text files

The text editor **`vi`** is a standard component of the Linux operating system and was created in 1976. While intimidating at first, this can be a very helpful editor when working with your files.

Key	Description
Arrow keys	Move around the file
`i`	Editing mode. This is great for preventing accidental changes to your file!
`esc`	Exit editing mode
`:q`	Quit – this will produce a warning if you made changes.
`:q!`	Quit the file and discard changes
`:w filename`	Save to `filename`
`:w! filename`	Overwrites to `filename`
`:wq`	Save and exit the file
`:colorscheme[space][tab]`	Get a selectable list of colour themes
`:color <theme>`	Directly select a colour theme

2.2.1 Open up the file in **`vi`** and delete the header of the file **HinfKW20_features_copy.txt**. Save it and compare file sizes with the original.

```
vi HinfKW20_features_copy.txt
      Move around the file with arrow keys, ctrl and shift
      i                                    # Enter edit mode
      Remove some of the header line with [backspace]
      [Esc]                                # Exit editing mode
      dd                                   # delete the entire header line
      :wq                                  # Save changes and quit vi
      ll
```

2.3.0 Concatenate files directly with `cat`

The **`cat`** command is useful when working with multiple files (like sequencing data) that you'd like to combine into a single file. Likewise, header-less data sets can be combined without opening an editor, copying, pasting, saving, etc.

2.3.1 Concatenate the **HinfKW20_cds.fna** and **HinfKW20_rna.fna** files together to get all these sequences into a single file.

```
ll *.fna
cat HinfKW20_cds.fna HinfKW20_rna.fna > HinfKW20_all.fna
      # remember > means you'll be sending output to a file (not a program)

ll *.fna
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll *.fna
-rwxr-xr-x 1 mokca mokca 1875338 Nov 22 15:15 HinfKW20_cds.fna*
-rwxr-xr-x 1 mokca mokca 1853084 Nov 22 15:15 HinfKW20_genomic.fna*
-rwxr-xr-x 1 mokca mokca   33031 Nov 22 15:15 HinfKW20_rna.fna*
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ cat HinfKW20_cds.fna HinfKW20_rna.fna > HinfKW20_all.fna
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll
total 8612
drwxr-xr-x 1 mokca mokca    4096 Nov 22 18:59 ./
drwxrwxrwx 1 mokca mokca    4096 Nov 22 15:16 ../
-rw-r--r-- 1 mokca mokca 1908369 Nov 22 18:59 HinfKW20_all.fna
-rwxr-xr-x 1 mokca mokca 1875338 Nov 22 15:15 HinfKW20_cds.fna*
-rwxr-xr-x 1 mokca mokca  466933 Nov 22 16:46 HinfKW20_features.txt*
-rwxr-xr-x 1 mokca mokca  466703 Nov 22 18:49 HinfKW20_features_copy.txt*
-rwxr-xr-x 1 mokca mokca 1853084 Nov 22 15:15 HinfKW20_genomic.fna*
-rwxr-xr-x 1 mokca mokca  665876 Nov 22 15:15 HinfKW20_protein.faa*
-rwxr-xr-x 1 mokca mokca   33031 Nov 22 15:15 HinfKW20_rna.fna*
-rw-r--r-- 1 mokca mokca 1481565 Nov 22 15:47 HinfKW20_sequences.tar.gz
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

2.4.0 Use `grep` to search through a file

We spent some time in Module 1 going over regular expressions when editing text. We saw how powerful it could be to use this language to search through our text files. grep is an essential command-line utility which has its origins in Unix but is now available for all Unix-like systems and exists in variant forms even in the Windows environment and many programming languages. grep generally takes the form of:

```
grep -switches "searchPattern" filePath
```

Switches	Description
-E	Use full regular expressions as defined in the bash shell
-P	Use Perl-compatible regular expressions
-i	Match without regard to case
-v	Invert the search
-c	Count the results in the file
-l	List only file names containing matches
-h	Hide filenames in the output

2.4.1 Count all the *unique* lines containing occurrences of tRNA and rRNA from the **HinfKW20_features_copy.txt** file using the **`grep`** command and in doing so, explore the functionality of the **>** and **>>** redirectors.

```
ls -l *.txt
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll *.txt
-rwxr-xr-x 1 mokca mokca 466933 Nov 22 16:46 HinfKW20_features.txt*
-rwxr-xr-x 1 mokca mokca 466703 Nov 22 18:49 HinfKW20_features_copy.txt*
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ grep -ci "tRNA" HinfKW20_features_copy.txt
67
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ grep -ci "rRNA" HinfKW20_features_copy.txt
38
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

2.4.2 Redirect the output of **`grep`** to a file using the **>** redirectors. Notice how the output of the grep command looks? The output highlights in red all occurrences of our search pattern, even if they should appear on multiple lines! Are we retrieving only **tRNA** features?

```
grep -in "tRNA" HinfKW20_features_copy.txt  # Note the output!
grep -i "tRNA" HinfKW20_features_copy.txt > HinfKW20_tRNA_features.txt
ls -l *.txt
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ grep -i "tRNA" HinfKW20_features_copy.txt > HinfKW20_tRNA_features.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll
total 8628
drwxr-xr-x 1 mokca mokca    4096 Nov 23 10:19 ./
drwxrwxrwx 1 mokca mokca    4096 Nov 22 15:16 ../
-rw-r--r-- 1 mokca mokca 1908369 Nov 22 18:59 HinfKW20_all.fna
-rwxr-xr-x 1 mokca mokca 1875338 Nov 22 15:15 HinfKW20_cds.fna*
-rwxr-xr-x 1 mokca mokca  466933 Nov 22 16:46 HinfKW20_features.txt*
-rwxr-xr-x 1 mokca mokca  466703 Nov 22 18:49 HinfKW20_features_copy.txt*
-rwxr-xr-x 1 mokca mokca 1853084 Nov 22 15:15 HinfKW20_genomic.fna*
-rwxr-xr-x 1 mokca mokca  665876 Nov 22 15:15 HinfKW20_protein.faa*
-rwxr-xr-x 1 mokca mokca   33031 Nov 22 15:15 HinfKW20_rna.fna*
-rw-r--r-- 1 mokca mokca 1481565 Nov 22 15:47 HinfKW20_sequences.tar.gz
-rw-r--r-- 1 mokca mokca    8450 Nov 23 10:19 HinfKW20_tRNA_features.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

2.4.3 Append the output of **grep** to a file using the **>>** redirectors. Rather than overwriting our destination file, we can append to our **HinfKW20_tRNA_features.txt** using the correct redirection symbol. Then we'll rename the file with the **mv** command.

```
grep -i "rRNA" HinfKW20_features_copy.txt >> HinfKW20_tRNA_features.txt
mv HinfKW20_tRNA_features.txt HinfKW20_RNA_features.txt
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ less HinfKW20_tRNA_features.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ grep -i "rRNA" HinfKW20_features_copy.txt >> HinfKW20_tRNA_features.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ mv HinfKW20_tRNA_features.txt HinfKW20_RNA_features.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll
total 8628
drwxr-xr-x 1 mokca mokca    4096 Nov 23 10:22 ./
drwxrwxrwx 1 mokca mokca    4096 Nov 22 15:16 ../
-rw-r--r-- 1 mokca mokca   12567 Nov 23 10:22 HinfKW20_RNA_features.txt
-rw-r--r-- 1 mokca mokca 1908369 Nov 22 18:59 HinfKW20_all.fna
-rwxr-xr-x 1 mokca mokca 1875338 Nov 22 15:15 HinfKW20_cds.fna*
-rwxr-xr-x 1 mokca mokca  466933 Nov 22 16:46 HinfKW20_features.txt*
-rwxr-xr-x 1 mokca mokca  466703 Nov 22 18:49 HinfKW20_features_copy.txt*
-rwxr-xr-x 1 mokca mokca 1853084 Nov 22 15:15 HinfKW20_genomic.fna*
-rwxr-xr-x 1 mokca mokca  665876 Nov 22 15:15 HinfKW20_protein.faa*
-rwxr-xr-x 1 mokca mokca   33031 Nov 22 15:15 HinfKW20_rna.fna*
-rw-r--r-- 1 mokca mokca 1481565 Nov 22 15:47 HinfKW20_sequences.tar.gz
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

```
less HinfKW20_RNA_features.txt
ls -l *.txt
```

2.4.4 How do we ensure we are counting only tRNA or rRNA features from our file? Based on our earlier **grep** output, it appears that feature types will either occur at the beginning of a line or in the second column. Here we can turn to Perl-compatible regular expressions with the **-P** switch. Remember what we learned earlier in **Module 1**?

Stick with the -P! By using PCRE we are able to use meta characters like tabs (\t) and spaces (\s)! There are other methods for utilizing tabs as well but they can be a little more complex. Using PCRE opens up additional features not provided by basic regular expressions (default) or even extended (-E) regular expressions.

2.5.0 Count words and lines with `wc`

Whenever you want to quickly investigate a file, the output of a command, or track how a file might be updating from another process, you can use the **wc** (**w**ord **c**ount) command to check the number of words, lines, or characters. This command generally takes the form of:

```
wc -switches filePath
```

Switches	Description
-l	Count the number of lines
-w	Count the number of words
-m	Count the number of characters

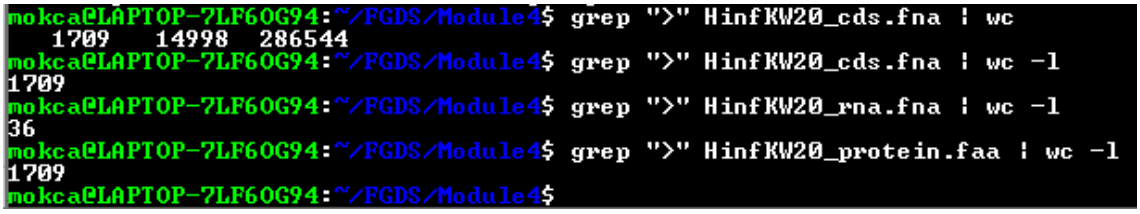
2.5.1 Count the number of words and lines in **HinfKW20_cds.fna** using the `wc` command.

```
less HinfKW20_cds.fna       # Take a quick look at the file
wc HinfKW20_cds.fna         # This will output lines, words, characters
```

2.5.2 Count and compare the number of features in the **HinfKW20_cds.fna**, the **HinfKW20_rna.fna**, and the **HinfKW20_protein.faa** files by combining the `grep` command and the `wc` command using `|`. The pipe symbol (`|`) redirects output from one command to the next, taking the place of input (ie `filePath` in this case).

Note that all fasta file entry headers start with the > symbol and that that grep -c ">" filename would also have worked here, but we wouldn't have gotten to see how `|` can be used to combine commands.

```
less HinfKW20_cds.fna
grep ">" HinfKW20_cds.fna
grep ">" HinfKW20_cds.fna | wc
grep ">" HinfKW20_cds.fna | wc -l
grep ">" HinfKW20_rna.fna | wc -l
grep ">" HinfKW20_protein.faa | wc -l
```



2.6.0 Quickly extract from tabular data with `cut`

When you generate data that might be in a tabular format, you may wish to look at specific columns or change the delimiter of the file (csv versus tsv, etc.). This may be after a series of commands or as part of a series where you wish to reformat or isolate output on the fly. The `cut` command takes the form of:

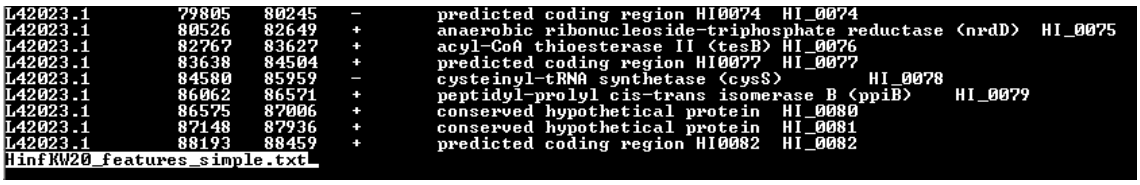
```
cut -switches parameters filePath
```

Key	Description
`-f, --fields=LIST`	Specify columns to return (comma-separated list)
`-d, --delimiter=DELIM`	Specify delimiter used to read/identify columns (tab is default)
`-c, --characters=LIST`	Specify range of characters to use (don't use on tabular data!)
`--output-delimiter=STRING`	Specify the type of delimiter to use for resulting output

2.6.1 Generate a simplified gene features file that only contains essential information regarding where and what each CDS feature is using the `cut` command. Call this new file **HinfKW20_features_simple.txt**.

```
ls -l *.txt
grep -i "CDS" HinfKW20_features_copy.txt
grep -i "CDS" HinfKW20_features_copy.txt |
grep -i "CDS" HinfKW20_features_copy.txt |
  HinfKW20_features_simple.txt

less HinfKW20_features_simple.txt
ls -l *.txt
```



2.7.0 Sort your data with `sort`

You want to quickly **`sort`** your tabular data in ascending or descending order. Again, this could be pre-, post- or mid-pipeline. The **`sort`** command takes the form of:

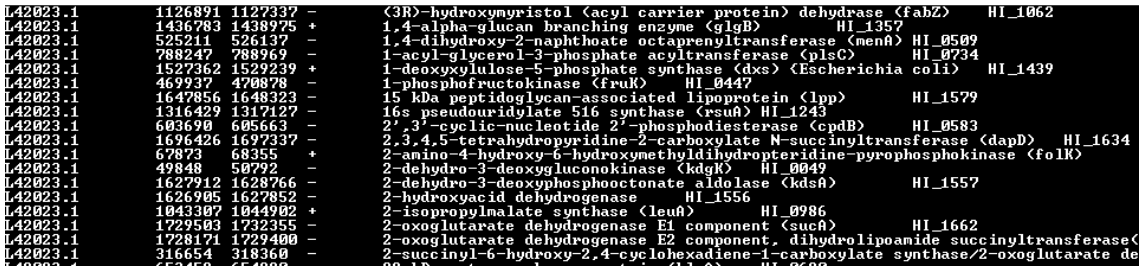
```
sort -switches parameters filePath
```

Switches	Description
`-k`	Specify specific column number to sort by
`-t`	Specify delimiter to separate columns
`-n`	Sort by numerical string value
`-r`	Sort in reverse order

2.7.1 Sort the output **HinfKW20_features_simple.txt** file by feature name rather than by their location in the genome using the **`sort`** command.

```
ll *.txt
                                                    >
  HinfKW20_features_simple_sortedbyproduct.txt

less HinfKW20_features_simple_sortedbyproduct.txt
ls -l *.txt
```



2.7.2 Sort the output **HinfKW20_features_simple.txt** file by column 3 using the **`sort`** command using both the string and numerical string values.

```
sort -k 3 HinfKW20_features_simple.txt | less
```

You may notice that there is a big difference between out two kinds of sorts. In the one instance, it's more of an alphabetical sort, not accounting for the context of non-leading 0 numbers. In the latter case, the **`-n`** switch attempts to give context to the column values by assuming these are numerical values!

3.0.0 Creating bash scripts and regulating access to files or programs

We are now going to learn how to combine commands into some simple scripts and how to regulate file access. This is where understanding your file system and directory tree really comes into play. Just like when you are in a Finder or File Explorer window, you can't open a file or run a program that is not in your current folder, but there are ways to manage this effectively. To begin, let's review some more helpful commands and files that are part of your Linux OS.

Command	Description
`bash`	Execution command (e.g. `bash script.sh`).
`sudo`	Super user permissions for system files (e.g. `sudo rm systemfile.txt`).
`$PATH`	Directories whose files or programs can be accessed from anywhere on your system.

`.bash_profile/`
 	
`.bashrc`
	Configuration script file in your root directory that is run every time you enter the shell. It is used to modify your $PATH (`.bash_profile` for MAC and `.bashrc` for LINUX).

3.1.0 Alter file permissions with `chmod`

When working with files, there are three general tasks you'd like to do with your files: read, write, and execute (if they are programs). In order use any of these options, a file must have those specific permissions enabled. Furthermore, every file keeps track of the general types of people (levels) that can access it: you (user), groups who own/share the file with you (group), and world-wide access or other users on the system (other).

Switches/Syntax	Description
`+, -, =`	Add, subtract, or set directly the permissions of a level
`u, g, o, a`	Level tags for user, group, other, or all
`r, w, x`	Permission tags to set for read, write, and execute
`4, 2, 1`	The octal values for read, write, and execute. These can be added to make specific combinations of permissions ie 6 -> read and write, vs 7 -> read, write, and execute and are set in a three-digit value to represent user/group/other

When using the **`chmod`** command, it takes the general form of

```
chmod options level=permissions,level2=permissions,level3=permissions filePath
```

or

```
chmod options 3-digit-octal filePath
```

Let's try some examples and see how it works in more detail.

3.1.1 **`chmod`** the permissions of all txt files to read for all levels and write only for the user.

```
ll *.txt
chmod u=rw,g=r,o=r *.txt
ll *.txt
```

3.1.2 **`chmod`** the permissions of all txt files with octal to read and write for user and group levels only and read access for other.

```
chmod 664 *.txt
ll *.txt
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll *.txt
-rw-r--r-- 1 mokca mokca  12567 Nov 23 10:22 HinfKW20_RNA_features.txt
-rwxr-xr-x 1 mokca mokca 466933 Nov 22 16:46 HinfKW20_features.txt*
-rwxr-xr-x 1 mokca mokca 466703 Nov 22 18:49 HinfKW20_features_copy.txt*
-rw-r--r-- 1 mokca mokca 118406 Nov 23 10:54 HinfKW20_features_simple.txt
-rw-r--r-- 1 mokca mokca 118406 Nov 23 11:02 HinfKW20_features_simple_sortedbyproduct.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ chmod u=rw,g=r,o=r *.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll *.txt
-rw-r--r-- 1 mokca mokca  12567 Nov 23 10:22 HinfKW20_RNA_features.txt
-rw-r--r-- 1 mokca mokca 466933 Nov 22 16:46 HinfKW20_features.txt
-rw-r--r-- 1 mokca mokca 466703 Nov 22 18:49 HinfKW20_features_copy.txt
-rw-r--r-- 1 mokca mokca 118406 Nov 23 10:54 HinfKW20_features_simple.txt
-rw-r--r-- 1 mokca mokca 118406 Nov 23 11:02 HinfKW20_features_simple_sortedbyproduct.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ chmod 664 *.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ ll *.txt
-rw-rw-r-- 1 mokca mokca  12567 Nov 23 10:22 HinfKW20_RNA_features.txt
-rw-rw-r-- 1 mokca mokca 466933 Nov 22 16:46 HinfKW20_features.txt
-rw-rw-r-- 1 mokca mokca 466703 Nov 22 18:49 HinfKW20_features_copy.txt
-rw-rw-r-- 1 mokca mokca 118406 Nov 23 10:54 HinfKW20_features_simple.txt
-rw-rw-r-- 1 mokca mokca 118406 Nov 23 11:02 HinfKW20_features_simple_sortedbyproduct.txt
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

3.2.0 Generating your first bash script

A bash script is a plain text file that carries a series of commands that can be executed by the command-line. All the calls we have been working with, can be included in a bash script to run and manipulate our files. Any commands from your bash script can be run directly in the command-line interface. We combine these commands into a single bash script to save time and "automate" our commands.

3.2.1 Create a new file titled **bash_script.sh** in a new directory titled **scripts**. This directory should be a subdirectory in the **Module4** directory. `sh` is an extension that is commonly applied to executable bash scripts. Make this file executable for all users with `chmod`. Note how the permissions on the file have changed.

```
mkdir scripts
cd scripts
vi bash_script.sh
      :wq
ll
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ mkdir scripts
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ cd scripts
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$ vi bash_scripts.sh
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$ ll
total 0
drwxr-xr-x 1 mokca mokca 4096 Nov 23 12:44 ./
drwxr-xr-x 1 mokca mokca 4096 Nov 23 12:44 ../
-rw-r--r-- 1 mokca mokca    0 Nov 23 12:44 bash_scripts.sh
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$ chmod a+rwx bash_scripts.sh
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$ ls -l
total 0
-rwxrwxrwx 1 mokca mokca 0 Nov 23 12:44 bash_scripts.sh
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$
```

3.3.0 Edit your bash script

Now that we have our bash script file, we can edit it again with `vi` to add some commands. We're going to include the following text in our bash script which will count the number of files in a directory and then count the number of lines, words, and characters in each file.

This script can then be run using the `bash` command. Recall that the `vi` command gets you into the script, the `i` key allows you to insert text into the file, the `esc` key cancels insert mode, and the `:wq` command sequence allows you to **w**rite (save) the script and **q**uit.

In many programming languages, everything to the right of a **#** (pound) sign within a script file is ignored (not interpreted nor executed). This feature allows us to add comments to our code to make more readable and easier to understand, without introducing bugs in our code

```
#! /bin/bash # this is called a shebang. Tells the system that this is a UNIX
   executable file

echo "This is how many files are in your directory:"
ls | wc -l # list files and pipes its output to count lines

echo "This is the number of lines, words, and characters in each file:"
# echo prints text into the console

for file in * # this is something called a for loop
do
wc -l $file # The $ refers to each file detected by out for loop
wc -w $file
wc -m $file
done
```

3.3.1 Open up vi, edit the file, insert the above code and save the file.

```
vi bash_script.sh
      i
      # copy and paste the above code into vi
      <esc>       # escape insert (writing mode)
      :wq         # save (write) changes and quit
less bash_script.sh
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$ ll
total 0
drwxr-xr-x 1 mokca mokca 4096 Nov 23 13:19 ./
drwxr-xr-x 1 mokca mokca 4096 Nov 23 12:44 ../
-rwxrwxrwx 1 mokca mokca  384 Nov 23 13:18 bash_script.sh*
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$ bash bash_script.sh
"This is how many files are in your directory:"
1
"This is the number of lines, words, and characters in each file:"
12 bash_script.sh
79 bash_script.sh
376 bash_script.sh
```

3.4.0 Run your bash script from other directories

Now try running the script from your **Module4** directory. This won't work unless you specify the absolute or relative path to your script. When you do specify the relative path, the script will run on your current directory. However, it won't be able to give you file sizes for any sub-directories like **scripts**.

3.4.1 Run your bash script from a different directory. You'll want to properly locate it by absolute or relative location.

```
cd ..
bash bash_script.sh              # Where do we keep the script?
bash scripts/bash_script.sh
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4/scripts$ cd ..
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ bash bash_script.sh
bash: bash_script.sh: No such file or directory
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ bash scripts/bash_script.sh
"This is how many files are in your directory:"
11
"This is the number of lines, words, and characters in each file:"
105 HinfKW20_RNA_features.txt
1475 HinfKW20_RNA_features.txt
12567 HinfKW20_RNA_features.txt
22557 HinfKW20_all.fna
36029 HinfKW20_all.fna
1908369 HinfKW20_all.fna
22145 HinfKW20_cds.fna
35434 HinfKW20_cds.fna
1875338 HinfKW20_cds.fna
3521 HinfKW20_features.txt
52198 HinfKW20_features.txt
466933 HinfKW20_features.txt
3520 HinfKW20_features_copy.txt
52177 HinfKW20_features_copy.txt
466703 HinfKW20_features_copy.txt
1709 HinfKW20_features_simple.txt
14998 HinfKW20_features_simple.txt
118406 HinfKW20_features_simple.txt
1709 HinfKW20_features_simple_sortedbyproduct.txt
14998 HinfKW20_features_simple_sortedbyproduct.txt
118406 HinfKW20_features_simple_sortedbyproduct.txt
22878 HinfKW20_genomic.fna
22885 HinfKW20_genomic.fna
1853084 HinfKW20_genomic.fna
9058 HinfKW20_protein.faa
22347 HinfKW20_protein.faa
665876 HinfKW20_protein.faa
412 HinfKW20_rna.fna
595 HinfKW20_rna.fna
33031 HinfKW20_rna.fna
wc: scripts: Is a directory
0 scripts
wc: scripts: Is a directory
0 scripts
wc: scripts: Is a directory
0 scripts
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$
```

3.5.0 Add scripts to your `$PATH` variable to access them anywhere

As you can see from our above example, when using the bash script we generated, it is important to know where the script is located. However, navigating to the location of a script when you are in a completely different directory can make your commands quite long. You can get around this by storing scripts close to the root or home directory OR you can make them part of your **`$PATH`** variable.

The **`$PATH`** environmental variable is an ***ordered*** list of paths that Linux will search through for executables when running a command. You do not need to include the executable itself but just the directory where it is located.

3.5.1 Add the scripts directory to your **`$PATH`** so that any script in the directory **scripts** can be run from anywhere on your system without having to specify the absolute or relative path to the script. You will need to edit your **`$PATH`** through your **.bash_profile** (Mac OS) or **.bashrc** (Windows/linux) to accomplish this.

```
cd ~
echo $PATH                    # See what your current $PATH setup includes
vi .bashrc                    # macOS: vi .bash_profile
      # Scroll to the bottom of the file
      i
            # Directories to add to path
            PATH="$PATH:[redacted]"
      <esc>
      :wq
```

3.5.2 Source your `.bash_profile` or `.bashrc` file and confirm it has been changed.

```
source .bashrc                # macOS: source .bash_profile
echo $PATH
cd FGDS/Module4
bash bash_script.sh
```

```
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ echo $PATH
/home/mokca/anaconda3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/mnt/c/Program Files/WindowsApps/CanonicalGroupLimited.Ubuntu20.04onWindows_2004.2021.825.0_x64__79rhkp1fndgsc:/mnt/c/Program Files (x86)/Common Files/Oracle/Java/javapath:/mnt/c/WINDOWS/system32:/mnt/c/WINDOWS:/mnt/c/WINDOWS/System32/Wbem:/mnt/c/WINDOWS/System32/WindowsPowerShell/v1.0:/mnt/c/WINDOWS/System32/OpenSSH:/mnt/c/Program Files/Intel/WiFi/bin:/mnt/c/Program Files/Common Files/Intel/WirelessCommon:/mnt/c/Program Files/MiKTeX/miktex/bin/x64:/mnt/c/Program Files (x86)/Intel/Intel(R) Management Engine Components/DAL:/mnt/c/Program Files/Intel/Intel(R) Management Engine Components/DAL:/mnt/c/Users/mokca/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/mokca/AppData/Local/GitHubDesktop/bin:/mnt/c/Program Files/MiKTeX/miktex/bin/x64/pdflatex.exe:/snap/bin:/home/mokca/FGDS/Module4/scripts
mokca@LAPTOP-7LF6OG94:~/FGDS/Module4$ bash bash_script.sh
"This is how many files are in your directory:"
11
"This is the number of lines, words, and characters in each file:"
105 HinfKW20_RNA_features.txt
1475 HinfKW20_RNA_features.txt
12567 HinfKW20_RNA_features.txt
22557 HinfKW20_all.fna
36029 HinfKW20_all.fna
1908369 HinfKW20_all.fna
```

4.0.0 Running operations on a remote server

The last thing we are going to learn today is how to access data and perform operations on a remote server. Genomic data and analyses often occur on remote servers that have more storage and computing power than your personal computers. Here are some important commands for running operations on remote servers:

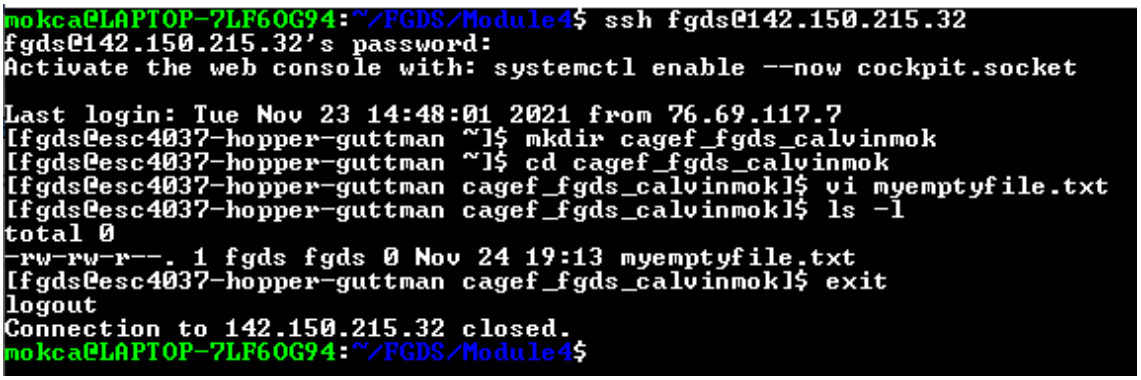
Switches/Syntax	Description
`ssh`	Start a secure remote shell connection (e.g. `ssh fgds@142.150.214.76 -p 24`)
`-p`	Specify the port to use
`sftp`	Start a file transfer remote shell connection (e.g. `sftp -oPort=24 fgds@142.150.214.76`)
`get`	Transfer from the server to your computer (e.g. `get HinfKW20_cds.fna`)
`put`	Transfer from your computer to the server (e.g. `put HinfKW20_cds.fna`)
`l`	Place before any command to run it on your computer rather than the server (e.g. `lcd`)
`quit`	Close the sftp connection
`scp`	Securely copy files to/from remote location (e.g. `scp HinfKW20_cds.fna fgds@142.150.214.76/home/HinfKW20_cds.fna`)

4.1.0 Connect and explore with `ssh`

We can use the secure shell protocol to communicate between two computers. This is more commonly known as **`ssh`** and uses an encrypted connection that is suitable for even on insecure networks. We'll briefly explore the basics of using **`ssh`** to connect, alter and explore another system.

4.1.1 Connect to our course server through an **`ssh`** connection and create a new directory for yourself with an empty file using **`mkdir`** and **`vi`**. Make sure to replace the directory name with **your own name** after "cagef_fgds". You will always disconnect from a server using the **`exit`** command.

```
ssh fgds@142.150.215.32
      PW = fall2022
mkdir cagef_fgds_your_name
cd cagef_fgds_your_name
vi myemptyfile.txt
      :wq
ls -l
exit
```



4.2.0 Send and retrieve files with `sftp`

The **`sftp`** or SSH File Transfer Protocol (aka Secure File Transfer Protocol) is another network protocol that uses **`ssh`** to facilitate file access, transfer, and management. This protocol uses the user-authenticated credentials required for an SSH connection.

4.2.1 Connect to our course server through a **`sftp`** connection and exchange content using the **`get`** and **`put`** commands. Note that all basic commands will apply to the server, but you can interact with your local computer by placing a "**`l`**" before the command when you're in a **`sftp`** connection.

```
cd cagef_fgds_your_name # replace with your own name
ls
get myemptyfile.txt   # "get" downloads myemptyfile.txt
lls # the preceding "l" points to your local computer (e.g. your laptop)
put HinfKW20_rna.fna  # "put" uploads HinfKW20_rna.fna to the server
ls
exit
```

That's it (for today)!

5.0.0 Class summary

That concludes our fourth lecture and introduction to the command-line tools and file structure. Next module we will take a closer look at installing programs in various ways through the command-line and contrast how some methods may be more appropriate to different needs of the data scientist. Altogether we've explored the following in this module:

- File and directory creation.
- Altering file content and file permissions.
- Running basic command-line tools for viewing and summarizing files.
- Creating your first bash script
- Setting up your `$PATH` variables
- Connecting to remote systems with `ssh` and `sftp`

5.1.0 Post-lecture assessment (10% of final grade)

Soon after this lecture, a homework assignment will be made available on Quercus in the assignment section. It will build on the ideas and/or data generated within this lecture. Each homework assignment will be worth 10% of your final mark. If you have assignment-related questions, please try the following steps in the order presented:

- Check the internet for a solution – read forums and learn to navigate for answers.
- Generate a discussion on Quercus outlining what you've tried so far and see if other students can contribute to a solution.
- Contact course teaching assistants or the instructor.

5.2.0 Suggested class preparation for Module 5

Next week we will begin exploring installations in the command-line interface and learning navigate our way through the spaghetti-string connections that result. There is nothing to help prepare for this except raw experience and lots of googling.

You will also have to install a package named Anaconda ahead of time. This will make your life much easier. Instructions will be sent through Quercus.

Appendix 1: Shell (Terminal) Commands

From Haddock & Dunn. 2011. Practical Computing for Biologists. Sinauer Assoc. Sunderland MA

Terminal operations are described in Chapters 4–6, 16, and 20. Many of the built-in `bash` shell commands are summarized here for quick reference. To get more information about a command and its options, type `man`, followed by the name of the command. If you are not sure which command applies, you can also search the contents of the help files using `man -k` followed by a keyword term.

Command	Description	Usage

`ls`	List the files in a directory
	Parameters that follow can be folder names (use * as a wildcard)
	`-a` Show hidden files
	`-l` Show dates and permissions
	`-1` List the file names on separate lines. Useful as a starting point for regexp into a list of commands
	`-G` Enable color-coding of file types
	`-F` Show a slash after directory names
		`ls -la`
		`ls -1 *.txt`
		`ls -FG scripts`
		`ls ~/Documents`
		`ls /etc`

`cd`	Change directory
	Without a slash, names are relative to the current directory
	With a preceding slash (/) names start at the root level
	Tilde (~/) starts at the user's home directory
	Two dots (..) goes "up" to the enclosing directory
	One dot refers to the current directory
	Minus sign goes to the previously occupied directory
	Use tab key (see below) to auto-complete partially typed paths
	Use backslash before spaces or strange characters in the directory name, or put the whole name in quotes
		`cd scripts`
		`cd /User`
		`cd ~/scripts`
		`cd My\ Documents`
		`cd 'My Documents'`
		`cd ../..`
		`cd ..`
		`cd -`

Command	Description	Usage
`pwd`	Print the working directory (the path to the folder you are in)	
↑	↑ key to step back through previously typed commands The cursor can be repositioned with the ← and → keys, and commands can then be edited Press *return* from anywhere in the line to re-execute. On OS X you can also reposition by *option*-clicking at a cursor location	
tab	Auto-complete file, folder, or script names at the command line	`cd ~/Doc` *tab*
`less`	Show contents of a file, page by page These commands also apply to viewing the results of `man` While `less` is running: `q` Quit viewing *space* Next page `b` Back a page `15 g` Go to line 15 `G` Go to the end ↑ or ↓ Move up or down a line `/abc` Search file for text abc `n` After an initial search, find next occurrence of the search item `?` Find previous occurrence of the search item `h` Show help for `less`	`less data.txt`
`mkdir`	Make a new directory (a new folder)	`mkdir scripts`
`rmdir`	Remove a directory (folder must be empty)	`rmdir ~/scripts`
`rm`	Remove file or files Use the `-f` flag to delete without confirmation (careful!) Use the `-r` flag to recursively delete the files in a directory and then the directory itself	`rm test.txt` `rm -f *_temp.dat`

`man`	Show the manual pages for a Unix command
	Use `-k` to search for a term within all the manuals
	The result is displayed using the `less` command above, so the same shortcuts allow you to navigate through
		`man mkdir`
		`man -k date`
		`man chmod`

Command	Description	Usage

`cp`	Copy file, leaving original intact
	Does not work on folders themselves
	Single period as destination copies file to current directory, using same name
		`cp test1.txt test1.dat`
		`cp temp ../temp`
		`cp ../test.py .`

`mv`	Move file or folder, renaming or relocating it Unlike `cp`, this does work on directories	`mv test1.txt test1.dat` `mv temp ../temp2`
`|`	Pipe output of one command to the input of another command	`history | grep lucy`
`>`	Send output of a command to a file, overwriting existing files Do not use a destination file that matches a wildcard on the left side	`ls -1 *.py > files.txt`
`>>`	Send output of a command to a file, appending to existing files	`echo "#Last line" >> data.txt`
`<`	Send contents of a file into command that supports its contents as input	`mysql -u root midwater < data.sql`

`./`	Represents the current directory in a path—the same location as `pwd`
	Trailing slash is optional
	Can execute a file in the current directory even when the file directory is not included in the `PATH`
		`cp ../*.txt ./`
		
		`./myscript.py`

`cat`	Concatenate (join together) files without any breaks. Streams the contents of the file list across the screen	`cat README` `cat *.fta > fasta.txt`
`head`	Show the first lines of a file or command Use the `-n` flag to specify the number of lines	`head -n 3 *.fasta` `ls *.txt | head`
`tail`	Show the last lines of a file or output stream Use the `-n` flag to specify the number of lines to show With a plus sign, skip that number of lines and show to the end. Use `-n +2` to show from the second line of the file to the end, skipping one header line	`tail -n 20 *.fta` `tail -n +3 data.txt`
`wc`	Count lines, words, and characters in an output stream or file	`wc data.txt` `ls *.txt | wc`
`which`	Show the location of executable files in the system path	`which man`

Command	Description	Usage

`grep`	Search for phrase in a list of files or pipe and show matching lines:
	`grep -E "searchterm" filelist`
	Often used in conjunction with piped output: `command | grep searchterm`
	Use quotes around search terms, especially spaces or punctuation like >, &, #, and others
	To search for tab characters, type ctrl V followed by tab inside the quotes
	Optional flags:
	`-c` Show only a count of the results in the file
	`-v` Invert the search and show only lines that do not match
	`-i` Match without regard to case
	`-E` Use full regular expressions
	Terms should be enclosed in quotes. Use `[]` to indicate a character range rather than the wildcards of Chapters 2 and 3
	General wildcard equivalents:
	`\s [[:space:]]`
	`\w [[:alpha:]]`
	`\d [[:digit:]]`
	`-l` List only the filenames containing matches
	`-n` Show the line numbers of the match
	`-h` Hide the filenames in the output	

`agrep`	Search for approximate matches, allowing insertions, deletions, or mismatched characters. (Must be installed separately.) See Chapter 21 Optional flags include: `-d ","` Use comma as delimiter between records `-2` Return results with up to 2 mismatches. Maximum is 8 mismatches `-B -y` Return the best match without specifying a number of mismatches `-l` Only list file names containing matches `-i` Match without regard to case	`agrep -d "\>" -B -y ATG seqs.fta` `agrep -3 siphonafore taxa.txt`

`chmod`	Change access permissions on a file (usually to make a script executable or Web accessible)
	First option is one of `u`, `g`, `o` for user, group, other
	Second option after the plus or minus is `r`, `w`, or `x`, for read, write, or execute. Can also use binary encoding as explained in Appendix 6
		`chmod u+x file.pl`
		`chmod 644 myfile.txt`
		`chmod 755 myscript.py`

Command	Description	Usage
`set`	Show environmental variables, including functions that have been defined	
`$HOME`	The environmental variable containing the path user's home directory	`echo $HOME` `cd $HOME`
`$PATH`	The user's `PATH` variable, where the directories to search for commands are stored	`export PATH=$PATH:/usr/local/bin`
`nano`	Invoke the text editor. Control key sequences include: ctrl X — Exit `nano` (will be prompted to save) ctrl O — Save file without exiting ctrl Y — Scroll up a page ctrl V — Scroll down a page ctrl C — Cancel operation ctrl G — Show help and list of commands	`nano filename.txt`
ctrl C	Interrupt the current process	

`sort`	Sort lines of a file
	`-k N` — Sort using column number *N* instead of starting at the first character. Columns are delimited by a series of white space characters
	`-t ","` — In conjunction with –k, use commas as the delimiter to define columns
	`-n` — Sort by numerical value instead of alphabetical
	`-r` — Sort in reverse order
	`-u` — Return only one unique representative from a series of identical sorted lines
		`sort -k 3 data.txt`
		`sort -k 2 -t "," F1.csv`
		`sort -nr numbers.txt`
		`sort A.txt > A_sort.txt`

`uniq`	Return a single line for each consecutive instance of that line in a file or output stream. To remove all duplicates from anywhere in the file, it must be sorted before being piped to the `uniq` command Use `-c` flag to return a count along with the repeated element	`uniq -c records.txt` `sort names | uniq -c`

Command	Description	Usage

`cut`	Extract one or more columns of data from a file
	`-f 1,3` Return columns 1 and 3, delimited by tabs
	`-d ","` Use commas as the field delimiter instead of tabs. Used in combination with –f
	`-c 3-8` Return characters 3 through 8 from the file or stream of data
		`cut -c 5-15 data.txt`
		`cut -f 1,6 data.csv`
		`cut -f2 -d ":" > Hr.txt`

`curl`	Retrieve the contents of a URL from over the network. URL should be placed in quotes. Without additional parameters, will stream contents to the screen For some Linux versions, `wget` offers similar functionality See `man curl` for ways to send user login information at the same time `-o` Set the name of the output file to save individual files for the data. See `#1` below `-m 30` Set a time out of 30 seconds `[01-25]` In the URL, substitute two digit numbers from 01 to 25 into the address in succession `{22,33}` `{A,C,E}` Substitute items in brackets into URL `#1` The substituted value, for use in generating the filename	`curl "www.myloc.edu" > myloc.html` `curl "http://www.nasa.gov/weather[01-12]{1999,2000}" -m 30 -o weather#1_#2.dat`
`sudo`	Run the command that follows as a superuser with privileges to write to system files	`sudo python setup.py install` `sudo nano /etc/hosts`
`alias`	Define a shortcut for use at the command line. To make persistent, add to startup settings file `.bash_profile` or equivalent	`alias cx='chmod u+x'`

`function`	Create a shell function—like a small script
	`$1` is the first user argument supplied after the command is typed
	`$@` is all the parameters—useful for loops as below
	Variable names are defined with the format `NAME=` with no spaces. They are retrieved with `$NAME`
	Save it in `.bash_profile` to make it permanent
		`myfunction() {`
		`  # insert commands here`
		`  echo $1`
		`}`

`;`	In a command or script, equivalent to pressing return and starting a new line	`date; ls`

Command	Description	Usage

`for`	Perform a `for` loop in the shell. Can be useful in the context of a function	`for ITEM in *.txt; do`
		`  echo $ITEM`
		`done`

`if`	An `if` statement in a shell function:
	`if [ test condition ]`
	`then`
	`  # insert commands`
	`else`
	`  # alternate command`
	`fi`
	Comparison operators are `eq` for equals, `lt` for less than and `gt` for greater than
		`if [ $# -lt 1 ]`
		`then`
		`  echo "Less than"`
		`else`
		`  echo "greater than 1"`
		`fi`

`` ` ` ``	Backtick symbols surrounding a command cause the command to be executed and then substitute the output into that place in the shell command or script	`cd `which python`/..` `nano `which script.py``
`host`	Return IP number associated with a hostname, or the hostname associated with an IP address, if available	`host www.sinauer.com` `host 127.0.0.1`
`ssh`	Start a secure remote shell connection	`ssh lucy@pcfb.org`
`scp`	Securely copy files to or from a remote location	`scp localfile user@host/path/remotefile` `scp user@host/home/file.txt localfile.txt`
`sftp`	Start a file transfer connection to a remote site. The prompt changes to an ftp prompt, at which the following commands can be used: `open` From the prompt, open a new `sftp` connection `get` Bring a remote file to the local server `put` Place a local file on the remote system `cd` Change directory on the remote server `lcd` Change directory on the local machine `quit` Exit the `sftp` connection	`sftp user@remotemachine`

`gzip`
`gunzip`
`zip`
`unzip`
	Compress and uncompress files	`gzip files.tar`
		`gunzip files.tar.gz`
		`unzip archive.zip`

`tar`	Create or expand an archive containing files or folders `-cf` Create `-xvf` Expand `-xvfz` Expand and uncompress `gzip`	`tar -cf archive.tar ~/scripts` `tar -xvfz arch.tar.gz`

Command	Description	Usage
`&`	When placed at the end of a command, runs it in the background	
`ps`	Show currently running processes. Flags controlling the output vary greatly by system. Usually a good starting point is `-ax`. See `man ps` for more	`ps -ax | grep lucy`
`top`	Show current processes sorted by various parameters, most useful of which is processor usage `-u`	`top -u`
`kill -9`	Terminate a process emphatically, using its process ID. Retrieve PID from the `ps` or `top` command	`kill -9 5567`
`killall`	Terminate processes by name	`killall Firefox`
`nohup`	Run command in background and don't terminate it when logging out or closing the shell window Use in this odd format shown, to prevent program output to cause the command to quit	`nohup command 2> /dev/null < /dev/null &`
ctrl Z	Suspend the operation to move it into the background or perform other operations	
`jobs`	Show backgrounded or suspended jobs, won't show normal active processes	
`bg`	Move a suspended process into the background. Optional number after it in the format `%1` will specify the job number	

`apt-get`
`yum`
`rpm`
`port`
	Package installers for various Unix distributions. Search for and install remote software packages. Typically used with `sudo`	`sudo apt-get install agrep`
		`yum search imagemagick`