Further Readings for Week 1

Below are some additional commonly used commands that were not covered in class but you may find helpful and want to explore further:

`whatis`

It displays one-line description for commands, handy tool for getting a quick overview of a command.

```
$ whatis whatis
whatis (1)           - display one-line manual page descriptions

$ whatis grep
grep (1)             - print lines that match patterns
```

`zcat`

It is similar to `cat` but works on gzip compressed files. Most file formats like fasta, fastq, gff and so on that Bioinformaticians commonly work with are gzipped for efficient storage. This command is useful to look into the contents of compressed files without having to uncompress them.

```
$ zcat Sequences/Sample_a/timepoint_1.fastq.gz | head
@Read_1
CAGGAAAGTCCCTACACTTTAACGGTGAGGCTAGAGATATGCCCGATACG
+
@D<<?DA;D?;=<?C;@>B;@:?<=?D>??@A@><:BA>=BB@=C;:<?:
@Read_2
ATTCATCGTGGCCAATCGGCCGGCGACGGTATGAGGTCTCAAGGAGTCCG
+
@;@@;?;:?C::BC@<:>D@A<<A;?:B<DC><D>C<?B>A@CC<C?CD?
@Read_3
GGGCAGACTGGCGCACGTGATCTTGAGGTCGGTGAGTGACTTTAGTAGTA
```

`zgrep`

It is similar to `grep` but works on gzip compressed files. `zgrep` invokes `grep`, so all the options that work with it also work with `zgrep`. Similar to the above, it is very useful for searching for patterns within compressed files without having to uncompress them first.

```
$ zgrep "@Read*" Sequences/Sample_a/timepoint_1.fastq.gz
@Read_1
@Read_2
@Read_3
@Read_4
@Read_5
@Read_6
@Read_7
```

```
@Read_8
@Read_9
@Read_10
```

`grep` - Other commonly used options

- `grep -r` - recursively search for patterns in all subdirectories.

The below command looks for pattern "@Read_2" in all files under the given path "Sequences/Sample_c/".

```
$ grep -r "@Read_2" Sequences/Sample_c/
Sequences/Sample_c/timepoint_3.fastq:@Read_2
Sequences/Sample_c/timepoint_2.fastq:@Read_2
Sequences/Sample_c/timepoint_1.fastq:@Read_2
Sequences/Sample_c/timepoint_5.fastq:@Read_2
Sequences/Sample_c/timepoint_4.fastq:@Read_2
```

As mentioned in the class, note that we can specify glob patterns in the input file paths given to `grep`. The below command greps for pattern "@Read_2" in all files with ".fastq" extension in "Sample_c" or "Sample_d" subdirectories.

```
$ grep "@Read_2" Sequences/Sample_[cd]/*.fastq
Sequences/Sample_c/timepoint_1.fastq:@Read_2
Sequences/Sample_c/timepoint_2.fastq:@Read_2
Sequences/Sample_c/timepoint_3.fastq:@Read_2
Sequences/Sample_c/timepoint_4.fastq:@Read_2
Sequences/Sample_c/timepoint_5.fastq:@Read_2
Sequences/Sample_d/timepoint_1.fastq:@Read_2
Sequences/Sample_d/timepoint_2.fastq:@Read_2
Sequences/Sample_d/timepoint_3.fastq:@Read_2
Sequences/Sample_d/timepoint_4.fastq:@Read_2
Sequences/Sample_d/timepoint_5.fastq:@Read_2
```

In both the cases above for grepping multiple files, the output shows the filenames in which the pattern was found followed by the match.

It is also common to chain multiple `grep` commands to narrow down the pattern matches. For example, if we only want fruits that contain character "p" that is not consumed raw, we can pipe grep commands as below:

```
$ paste Research/Fruit/fruits.txt Research/Fruit/preparation.txt
Research/Fruit/scores.txt > Research/Fruit/concat_fruits.txt
$ head -5 demo/Research/Fruit/concat_fruits.txt
Apple   both    6
Mango   raw     9
Pear    both    7
Durian  raw     3
Lychee  raw     8
$ grep "p" Research/Fruit/concat_fruits.txt
Apple   both    6
```

```
pineapple       raw     3
eggplant        cooked  9
grapes  raw     3
cantaloupe      raw     2
pomegranite     raw     4
$ grep "p" demo/Research/Fruit/concat_fruits.txt | grep -v "raw"
Apple   both    6
eggplant        cooked  9
```

We will see more practical examples of this when we get to the lecture on regular expression.

- `grep -A <n>` - shows the line(s) with matching pattern plus "n" number of lines after the matching lines. This is helpful in matching patterns based on context. The below command prints the 3 lines after the header line containing "@Read_2":

```
$ grep -A 3 "@Read_2" Sequences/Sample_d/timepoint_1.fastq
@Read_2
AATATTTCGTTTTACGTATCTCCATACTTGTCCACTAGATCCAGGCAATG
+
=:?;=:=AB?D<?ADA?=@;?B:@@@>@BBC==><AD@A@C@B<=D<>=;
```

Similarly, `grep -B <n>` prints "n" number of lines before the line with matching pattern.

- `grep -f <file>` - if there are multiple patterns to match, we can create a file with patterns and use that with grep.

```
$ echo -e "Apple\nTomato" > patterns.txt
$ cat patterns.txt
Apple
Tomato
$ grep -f patterns.txt demo/Research/Fruit/concat_fruits.txt
Apple   both    6
Tomato  raw     5
```

`tr`

Used to translate or delete characters. It takes input from stdin and writes to stdout. For example, to convert a tab-separated file into a comma-separated file, we can:

```
$ echo -e "Item1\tItem2\tItem3" > items.tsv
$ cat items.tsv
Item1   Item2   Item3
$ cat items.tsv | tr "\t" ","
Item1,Item2,Item3
```

`awk` and `sed`

`awk` and `sed` are powerful tools that can be used to manipulate text files. Both these tools work line-by-line by default. There are creative ways to use these languages to create succinct scripts for complex tasks due to the vast number of options they provide. You can take a look at the cheatsheets for `awk` here and `sed` here to get to know the available functionalities.