**Advanced Unix/Linux and Shell Scripting**

1. Redirection
    a. Redirecting the output of a command to a file can be extremely useful.
        i. For instance, we can use **grep** and redirect the output to a new file.
            1. **grep** normally outputs to the screen.
        ii. If we use the file "Sample_Data.csv" from last week, we can:
            1. Move the header to a new file:
                a. `grep catalognumber Sample_Data.csv > Alaska_Samples.csv`
            2. Get all of the rows from Alaska and add them to the new file:
                a. `grep Alaska Sample_Data.csv >> Alaska_Samples.csv`
2. Piping
    a. We can use piping to send the output from one program to another program.
        i. The pipe operator is |.
        ii. For example, say we want a list of the location where samples were collected from "Sample_Data.csv".
            1. We can use **cut** to get that column.
                a. `cut -d "," -f 11 Sample_Data.csv`
            2. We can pipe it into **sort** to sort the column.
            3. We can pipe it into **uniq** to reduce it to the unique locations.
            4. We can use **tail** to get rid of the "stateprovince" from the header.
                a. `tail -n +2 Sample_Data.csv | cut -d "," -f 11 | sort | uniq`
            5. Finally, we can redirect all of this into a file if we want.
                a. Just add `> stateprovince.txt`
3. Other command-line utilities
    a. **cat**: concatenate and print files.
    b. **wc**: line, word, and byte count of a file.
    c. **sort**: sort the contents and print to screen.
    d. **uniq**: show only unique lines (has to be sorted first).
    e. **file**: use this to see if the file has Windows line breaks (CRLF).
    f. **cut**: extract columns from a file. Use -d to specify the delimiter and -f to specify the column numbers.
    g. **tr**: substitute characters in a text string. Ex: echo "GATTCTC" | tr T U
    h. **grep**: search for lines in a file that match a text query.
    i. **sed**: a very useful utility for manipulating text files (especially replacing characters).
    j. **awk**: another utility for searching and processing of text files.
    k. **gzip** and **gunzip**: Utilities for compressing and decompressing gz files.
    l. **tar**: Compress and decompress "tar" or "tar.gz" archives. Google it to see which arguments you should use.
4. Wildcards
    a. In Linux, the ***** will substitute for any other string of characters.
        i. If you want to decompress all ".gz" files in a folder, for example, you could type: `gunzip *.gz`
5. Basic scripts
    a. Just a list of commands in the order you would like to run them.

    b. This sort of script can be extremely valuable in automating tasks and in keeping a record of you work.
    c. Your scripts might be made publicly available when you publish a paper.
6. Scripts for running jobs on a cluster
    a. See this website https://hpc.uidaho.edu/general/Workshops/Cluster_computing.html.
    b. Our computing cluster uses Slurm to schedule jobs.
7. Advanced shell scripting
    a. A script can accept arguments.
    b. A script can contain loops.
    c. A script can use if statements.
    d. Other programming techniques can also be used in scripts.
    e. A good practice is to keep scripts simple and clear.
        i. Use a more advanced programming language for the complicated stuff.
8. Some shell script examples.
    a. See the next page.

**Shell scripts:**

How to write a shell script, examples:

A script that lists the files in the home directory:

```
#!/bin/bash
ls ~/
```

Just a basic shell script that lists the fasta and fastq files in the current folder:

```
#!/bin/bash

echo "Fasta Files:"
ls *.fasta

echo "Fastq Files:"
ls *.fastq
```

A script to split a fasta file into individual records (assuming the fasta file has no line breaks in the sequences): [An example of using arguments]

```
#!/bin/bash

# This script takes 1 fasta file and splits it into individual records.
# This won't work if the sequences contain line breaks.
# This takes 1 argument, the name of the fasta file.

split -l 2 $1 out_fasta
```

Add a ".fasta" to each of these new files: [An example of a for loop]

```
#!/bin/bash

for filename in $(ls $1*)
do
      echo $filename
      cp $filename $filename.fasta
done
```