# Session #4

# Take Home Assignment #3

- Feedback
- Google Form results

The GSBS Bootstrappers

# Small Scripts

- Scripts will allow you to run multiple commands at the same time – great for analysis on multiple datasets
- You can incorporate any command line tools (Python, Perl, samtools, bedtools)
- Better way to keep track of your analysis pipelines especially when it comes time to publish.

# Small Scripts

```
#!/bin/bash

Command1

Command2

Command3
```

# Executing Scripts

- Make sure that your script is executable:

```
chmod +x Script.sh
```

- You can then execute the script by typing the path + script name

```
/Home/Path/to/Script.sh

./Script.sh

~/Path/to/Script.sh
```

The GSBS Bootstrappers

# Variables

- Variable allow you to store values (number, word, path, etc.)

- You define a variable as:

```
x=1
y=hello
Data=/home/jm36w/experiment1
z="hello world"
```

- You refer to a defined variable as:

```
$x
$y
$Data
$z
```

# Printing Variables

- You can print variable's value with echo:

```
j= "hello world"
echo $j
```

# Using Variables with Other Commands

1. 
```
wc -l file.txt
```

2. 
```
F=file.txt
wc -l $F
```

3. 
```
F=file
wc -l $F".txt"
```

# Using Variables with Other Commands

1. 
```
grep “green” File.txt
```

2. 
```
color=green
grep “$color” File.txt
```

# Passing Arguments

- If you want greater flexibility with your script, you can utilize command line arguments

```
./sample-script.sh Arg1 Arg2
```

# Passing Arguments

- You can then refer to these arguments in your script as $1 for Arg1, $2 for Arg2 etc

```
wc -l $1
echo $1

wc -l $2
echo $2
```

# Exercise Set #6

# Using Variables with AWK

- Since AWK is its own language, it has its own notation for variables
- Therefore to use Bash variables within AWK, you must be very careful with your syntax

# Using Variables with AWK

- You can refer to any variable OUTSIDE of the single quotes using $variable notation

```
F=sample-file.txt
awk '{print $1}' $F
```

# Using Variables with AWK

- You can refer to any variable INSIDE of the single quotes using '$variable' notation

- Think of it as using single quotes to leave awk interpreter

```
c=green
awk '{if ($1 == "'$c'") print $0}' File.txt
```

# More Fun with awk

- More helpful awk one liners that I have found useful for bioinformatics analysis
- Power of awk is not limited to what is listed here!

# awk- Comparing Files

- `awk 'FNR==NR {x[$1];next} ($2 in x)' File1.txt File2.txt`

Store column 1 of file 1 in memory

For each line in file 2, if column 2 is in memory print line

# awk- Printing the Nth Line

- `awk '{if (NR==1000) print $0' file`

Print the 1000th line

# awk- Printing Every Nth Line

- `awk 'NR%10==1' file`

Print every 10th line starting with line 1

If mod 10(line number) equals 1 print line

- `awk '{if (NR%10==1) print $0}' file`

# awk- Replacing Characters

- `awk '{gsub(/foo/,"bar"); print}' File1.txt`

Replace all instances of foo with bar and print out everything to the terminal

# awk- BEGIN & END Blocks

- `BEGIN {}` – Execute this code first before cycling through each line
- `END {}` – Execute this code after cycling through each line
- `awk 'BEGIN {First Stuff} {Main Stuff} END {End Stuff}'`

# awk- Maximum/Minimum of Column

- `awk 'BEGIN {max = 0} {if ($3>max) max=$3} END {print max}'`

Print the maximum value of column 3

- `awk 'BEGIN {min = 100} {if ($3 < min) min=$3} END {print min}'`

Print the minimum value of column 3

The GSBS Bootstrappers

# awk- Sum of a Column

- `awk 'BEGIN {SUM=0} { SUM += $1} END { print SUM }' File1.txt`

Print the sum of column 1

# awk- Print a Header

- `awk 'BEGIN {print “Name \t Age”{ print $1 “\t” $2 }' File1.txt`

Print a tab delimited header before the data