In Class Demo: Bash scripting

```
> echo $PATH          # Prints the folders in which Bash knows to look to execute a command

> cd Desktop/CSCI3308 # cd change directory from ~/ to ~/Desktop/CSCI3308

> pwd                 # pwd prints the current working directory

> PATH=$PATH:/Users/anni/Desktop        # Add /Users/anni/Desktop to the PATH
                                        variable

> vim tmp.sh          # Opening a file with the vim text editor
                      # If the file does not exist, vim will create it

                      # vim has two modes, insert and command, type "i" to insert, hit esc to
                      use command mode

#!/bin/sh             # Putting this at the beginning of a bash script tells the system that it is a
                      shell script

ls -a                 # ls list all of the files in the current director; the 'a' option lists all files
                      including hidden files

:w                    # Type :w to write changes to a file
:q                    # In vim, type :q to quit

> tmp.sh              # This command will not work; we have to specify the directory
> ./tmp.sh            # '.' means current directory

> chmod +x tmp.sh           # Sometimes we need to change the permissions of a file
                            # chmod +x gives everyone executable permission (r=read,
                            w=write, x=execute)

> echo $USER          # USER is another environment variable; it just exists, we don't have to
                      create it

> pwd                 # Don't confuse the commands and environment variables; pwd is a
                      command, but capital PWD is an environment variable
> echo $PWD

> day=Wednesday                     # This is how we assign values to variables
> echo "Today's day is $day"        # To print the content of variable day we need to
                                    use $

> day=5                             # Variables are untyped, we can put anything in
                                    variables
> echo "Today's day is $day"
```

```
> a=1+2     # Because it's untyped, bash doesn't know to evaluate this arithmetic statement
> echo $a   # Output: 1+2

> let a=1+2       # let can be used to tell Bash that these are numbers
> echo $a         # Output: 3

> let a=hello
> echo $a               # Output: 0; "hello" is a string but Bash treats it as a number
                        because we didn't specify that "hello" is a string
> let a="hello world"   # Use quotes for strings;  this will produce an error because it is
                        not an arithmetic expression and Bash does not know how to
                        evaluate it

> a="hello world"       # You can assign strings without the keyword let
> echo $a               # Output: "hello word"

> a=$((1 + 2))          # For more complicated arithmetic we can use $((…))
> echo $a               # Output: 3

> a=$(ls -l)            # For command substitution we can use $(…)
> echo $a               # Output: -rw-r--r--1 username group filesize date filename
                        # Note how there are no newlines in this output
> echo "$a"             # Putting quotes around the variable will give us newlines

> a=$(expr 1 + 2)       # Again, this is command substitution $(expr expr1 expr2 expr3)
                        # expr is like a small command-line calculator
> echo $a               # Output: 3

> a=`ls -l`             # This is another way to do command substitution
> echo $a

> unset a         # Literally unset the variable; it still exists but is empty
> echo $a         # Output: -rw-r--r--1 username group filesize date filename

> a=2
> readonly a      # readonly makes the variable read only
> a=3             # This produces an error because the variable is read only
                  # Note, it is difficult to unset a read only variable
                  # https://stackoverflow.com/questions/17397069/unset-readonly-variable-
in-bash


> set             # List all variables (environment and shell) and functions
```