Bash Scripting 101

#!/bin/bash - #! is called a shebang

echo $PATH to display what executable paths are available in your environment (can be changed/added)

Place script inside of /home/user/bin for script to be executable anywhere on command line

Scripts must have proper executable permissions to run

To run a script not located in bin reference the script path /home/user/myscript or ./myscript

A script can just be a set of Linux Commands.

Example: myscript
#!/bin/bash
echo “Hello World”
ls
pwd

Summary: Script will Display Hello World then list contents of directory then display current working directory.
Example Script From lesson two:

```
#!/bin/bash
#List all contents in a directory and write the output to a file named dir_list.txt
location=$1
filename=$2

if [ -z "$location" ]
then
echo "please provide location"
exit
fi

if [ -z "$filename" ]
then
echo "please provide filename"
exit
fi

ls $location > $filename
echo "Script is complete and has indexed $location"
echo "############"
echo "Displaying contents of $location"
echo "############"
cat $filename
```

Variables & Arguments

(()) used to interpret arithmetic operations
$# displays number of arguments passed to the script
if (($# == 0)) then echo “No arguments have been passed to the script” fi
{} around variables names such as ${variable} are used to unambiguously identify variables.

- $ VARIABLE=hello
- displays: variable: hello

- echo Variable: $variable1234
- displays nothing since $variable1234 is not a variable
- echo ${variable}1234 interprets the variable as is and ads on 1234
- displays: hello1234

Conditions In Bash Scripting

```
#!/bin/bash
#conditions in bash scripting

# [ means test [[ means new test using [ is same as
using the command "test" to evaluate.
file=name
if test -f $file
then
     echo "is a a file too"
fi
[ -f $file ] && echo "$file is a file"


#single bracket if statements refered to as "test"
brackets oldest and most compatibal "test"
#basic syntax have to quote strings cannot do file
globbing
emptystring=""
if [ -z "$emptystring" ]
then
    echo "string is empty"
fi

if [ "$emptystring" == "" ]
then
    echo "with single test brackets you must quote
your strings at all times"
fi

#flag conditions
#-gt = greater than
#-lt = less than
#-ge = greater than equal to
#-le = less than or equal to
#-eq = equal to
#-ne = not equalto
#-f = is file
```

```
#-d = is directory
#-l = is symlink

if [ 2 -gt 1 ]
then
     echo "yes 2 is greater than 1"
fi



#double test brackets [[ ]]
#[[ allows shell globbing which means an * will expand
to literally anything
#word splitting is prevented so you can omit placing
quotes around string variables but it's not best
practice

mystring=sammy
if [[ "$mystring" == *mmy* ]]
then
     echo "This determines if the string contains mmy
anywhere in it"
fi

if [[ $mystring == *[sS]a* ]]
then
     echo " (notice no quotes) this determines if the
string contains sa or Sa anywhere in it"
fi

#expanding files names using [[]]

if [ -a *.txt ] #returns true if there is one single
file in the current working directory that has .txt in
it
then
     echo "* with single test brackets expands to the
entire current working directory so it will error if
more than 1 file exists"
     echo "there is at least one file that ends with
.txt in the dir"
fi

if [[ -a *.txt ]] #with double brackets the * is taken
literally'
```

```
then
     echo "returns true only if there is a file name
*.txt (literally name"
fi


##double brackets allow for && and ||
## double brackets allow for regular expressions using
=~ not to be covered in this course
#&& is and for [[ but single test -a also works
#|| is for or and -o for single bracket also works


#double parenthesis (( )) used primarly for number
based condiations and allows use of >= operators
#Does not let you use flag condiations
#allows the use of && and || but not -a -o
#same as using the let command
```

remove /root/labackup when compelted
ssh root login

Loops

#!/bin/bash
#loops

#for arg in [list]
#do
commands
#done

#while [condtion]
#do
commands
#done

for file in `ls`
do
     echo $file >> filelist.txt

```
done



for line in `cat filelist.txt`
do
	echo $line

done

count=0
while [ $count -lt 10 ];
do
	echo Counter is $count
	let count=count+1
	echo $count;
done
```

Addusers Practice Script
Script will add and remove system users based off a listed text file.

```
#!/bin/bash

file=$1
action=$2

if [ -z  "$file" ]
then
	echo "Please enter a file with users"
	exit 0
fi

if [ -z "$action" ]
then
	echo "Please enter del or add"
	exit 0
fi

for user in `cat $file`
do
	if [ "$action" == "add" ]
```

```
    then
        echo adding user: $user
        useradd $user -m -ppassword
    fi
    if [ "$action" == "del" ]
    then
        echo deleting user: $user
        userdel -r $user
    fi

done
```

#end add users script

Functions and Case Statements

```
#!/bin/bash

action=$1

function first {
    echo "this is the first function"
}

function second {
    echo "this is the second fucntions"
}


case $action in
1)
    first
;; #each clause is terminated with ;;
2)
;;
*)
    echo "you have not picked a valid option"

esac
~
```

Interactive scripts with read

-t timeout if the input is not given within a certain time frame the script will move on

-n returns after reading n number of characters

-r Backslash does not act as an escape character but is instead considered part of the line

```
To Do Script
echo -n "What would you like to do: "

read -n2 todo

if [ `grep $todo todo.txt | wc -l` -eq 1 ]
then
      echo "Already in your list"
else

 echo $todo >> todo.txt

echo "added to your list"
fi
```