**Linux Review**

1. How do you know if you are in your home directory?
2. What can a script do?
3. How do you start the editor?
4. How do you include a comment in your script?
5. How do you run a script?
6. How do you change a script?
7. How do you put security on your script so that only you can run, see and edit it and others cannot do anything to it?

**Linux Scripts – Intro and IF**

Create the Start2 script:

#Start2 script

#{Your Name}

#This is a second example of a startup script

#Displays current date and time, number of users

#and current working directory

echo

echo “Date and Time:”

date

echo “Number of users on the system:”

who|wc –l

echo “Your current directory:”

pwd

echo

^X

Name the file start2

chmod 700 start2

sh start2 [return]

Echoline Script

This script requests the user to enter a line of text and then repeats it back word by word.

In the echo statement,

-e is used to enable the interpretation of backslash-escaped characters including the

\c option which is to suppress a trailing newline

Notice that the words in the sentence are broken out into 5 variables (w1, w2, w3, w4, and w5). Also notice that when these variables are used in the echo statement they have a $ in front of them so that the value of the variables is displayed and not the name of the variables.

***jpico*** [return]

Type the following:

#Echoline script

#{Your Name}

#this script reads and echoes multiple variables

#

echo –e “Enter a line of text: \c”

read w1 w2 w3 w4 w5

echo “The first word is: $w1”

echo “The second word is: $w2”

echo “The third word is: $w3”

echo “The fourth word is: $w4”

echo “The rest is: $w5”

^X

Name the file echoline

chmod 700 echoline

sh echoline [return]

Type This is a test of my script. [return]

Run it again and enter another longer sentence.

How could you change the script to repeat more words?

**If statements in LINUX scripts:**

If statements are used to control the flow of logic in a script so that only certain commands are executed if certain conditions are met. For example, if a student earns at least 400 points in a certain class, then they pass the class. If they do not earn at least 400 points, then they do not pass the class.

If statements have a condition portion, what to do if the condition is true and what to do if the condition is false. They also have a scope terminator – something to tell the script that the if statements has ended.

The condition portion is written in a variety of ways, but is typically comparing one thing against another in terms of equality or less than or greater than. Symbols used include

= for equal to, -eq for equal to, -lt for less than, -gt for greater than,

-le for less than or equal to, -ge for greater than or equal to.

See man expr for more discussion of expressions in Linux.

An if statement in a LINUX script file has the following format:

**if** condition

**then**

what to do if the condition is true

**else**

 what to do if the condition is false

**fi**

The word “then” is required for the “if” statement. The word “else” is optional.

fi is required and ends the “if” statement.

In the example below, the read statement is used with the –p option. This is the option that specifies that a prompt will be used. The text in quotes "Guess my favorite number: " is the prompt. When the user enters a number, that becomes the value of the variable num. As in the previous example, to tell the shell to use the value of the variable num, it is used with a $ in front of it.

read -p "Guess my favorite number: " num

Number Example:

echo "Welcome to the guess my favorite number game"

read -p "Guess my favorite number: " num

if [ $num = 7 ]

then

 echo $num "is my favorite number too!"

 echo "You win the prize!"

else

 echo "Although not my favorite,"

 echo "the" $num "is a cool number too!"

fi

uoas050@elinux:~> sh number

Welcome to the guess my favorite number game

Guess my favorite number: 7

7 is my favorite number too!

You win the prize!

uoas050@elinux:~> sh number

Welcome to the guess my favorite number game

Guess my favorite number: 9

Although not my favorite,

the 9 is a cool number too!

Fruit Example:

In the fruit example, the user is asked to enter their favorite fruit. Notice the if statement above in which the favorite number is not in quotes. Compare that to the if statement below in which the favorite fruit is in quotes. This is because the fruit is not numeric data.

**jpico fruit** [return]

Type the following in the editor:

echo "YOUR NAME Fruit Stand"

read -p "Enter your favorite fruit: " fruit

if [ $fruit = "apple" ]

then

 echo "The" $fruit "is my favorite fruit too!"

 echo "Free groceries for you for the day!"

else

 echo "Although not my favorite,"

 echo "the" $fruit "is good too!"

fi

^X

Name the file fruit

sh fruit [return]

Run the script twice – once with apple as the fruit and once with another fruit as the fruit:

uoas050@elinux:~> sh fruit

Dave's Fruit Stand

Enter your favorite fruit: apple

The apple is my favorite fruit too!

Free groceries for you for the day!

uoas050@elinux:~> sh fruit

Dave's Fruit Stand

Enter your favorite fruit: orange

Although not my favorite,

the orange is good too!

Status codes Example:

To demonstrate the next script, you first need to create a file named october

In this example, a status code is used. If the command is successfully completed, then the system returns a code of 0; otherwise, the code is not 0 (it is an error code of some kind if it is not 0)

Type the following in the editor:

if mv october nov

then

 echo “Move completed successfully – file moved.

 Status “ $?

else

 echo “Move completed unsuccessfully – file not

 moved. Status “$?

fi

^X

Name the file moveex

Run the script twice. If it worked correctly the first time, why doesn’t it work correctly the second time?

uoas050@elinux:~> sh moveex

Move completed successfully - file moved.

 Status 0

uoas050@elinux:~> ls –l

What do you notice?

uoas050@elinux:~> sh moveex

mv: cannot stat `october': No such file or directory

Move completed unsuccessfully - file not

 moved. Status 1

**Logical Operators in scripts**

AND is represented with &&

OR is represented with ||

Not is represented with !

Customer Service Example:

Customer service representatives receive a bonus if their calls handled are greater than 150 or if their calls closed are greater than 50; otherwise, they do not receive a bonus.

read -p "Enter the calls handled: " CallsHandled

read -p "Enter the calls closed: " CallsClosed

if [[ $CallsHandled -gt 150 || $CallsClosed -gt 50 ]]

then

 echo "You are entitled to a bonus."

else

 echo "You are only entitled to a bonus if the calls handled

exceeds 150 or calls closed exceeds 50."

fi

^X

Name the file bonus

Run the script

Enter 159 for calls handled and 40 for calls closed. Did you get a bonus?

Run it again with 140 calls handled and 40 calls closed. Did you get a bonus?

Run it again with 160 calls handled and 60 calls closed. Did you get a bonus?

Copy the file bonus to bonus2 and change the script to give a bonus when the calls handled are greater than or equal to 150 or if their calls are greater than or equal to 50.

Copy the file bonus to bonus3 and change the script to give a bonus when BOTH conditions are met.

Run the new script with the same three scenarios. Did you get the response you should?

Retire Example:

Write a script to determine if someone can retire. If the years of service is not less than 25, then they can retire, otherwise, a message is printed telling them they need 25 years of service to retire.

read –p “Years of Service: “ Years

if [[ ! $Years –lt 25 ]]

then

 echo “You can retire now.”

else

 echo “You will need 25 years to retire.”

fi

^X

Name the file retire

Run the script and respond 32. Run it again and respond 21.

**ELSE IF**

ElseIF - Using the elif Clause in an if Statement:

**Generic syntax:**

if *list*

then

 statements

elif *list*

then

 statements

else

 statements

fi

Example: Revisit the Guess my number script and give messages specifying if the number guessed is greater than or less than the favorite number if the number is not guessed. First, copy the number script to numberif and then edit it to something similar to the following:

echo "Welcome to the guess my favorite number game"

read -p "Guess my favorite number: " num

if [ $num = 7 ]

then

 echo $num "is my favorite number too!"

 echo "You win the prize!"

elif [ $num -gt 7 ]

 then

 echo "Although not my favorite,"

 echo "the" $num" is a cool number too!"

 echo "and is greater than my favorite number"

 else

 echo "Although not my favorite,"

 echo "the" $num" is a cool number and is less than my favorite

number"

fi

Run the script at least three times – once supplying the actual number, once supplying a number greater than the number and once supplying a number less than the number. Do you get the expected results?

Example: Create a script that indicates whether a company has achieved one of three scenarios – profit, loss, or breakeven.

read -p "Enter Sales Amount: " sales

read -p "Enter Costs: " costs

((net=$sales - $costs))

if [[ net -eq 0 ]]

then

 echo "Profit and Costs are equal - breakeven."

elif [[ net -gt 0 ]]

 then

 echo "Profit of: " $net

 else

 echo "Loss of: " $net

fi

Save the script as proflossbe.

Run the script inputting 5000 for Sales Amount and 6500 for Costs.

Run the script inputting 6500 for Sales Amount and 6500 for Costs.

Run the script inputting 6500 for Sales Amount and 5000 for Costs.

The results you should see are included below:

uoas050@elinux:~> sh proflossbe

Enter Sales Amount: 5000

Enter Costs: 6500

Loss of: -1500

uoas050@elinux:~> sh proflossbe

Enter Sales Amount: 6500

Enter Costs: 6500

Profit and Costs are equal - breakeven.

uoas050@elinux:~> sh proflossbe

Enter Sales Amount: 6500

Enter Costs: 5000

Profit of: 1500

**Nesting IF Statements**

In this example, a person is eligible to retire if their age is at least 60 with 10 years of service OR if they have 25 years of service regardless of age. Think about this one before entering the script. Sample results are included below:

uoas050@elinux:~> sh retirementstatus

Enter Age: 50

Enter Years of Service:10

In order to retire, you must be at least age

60

with 10 Years of Service or

have 25 years of service.

uoas050@elinux:~> sh retirementstatus

Enter Age: 62

Enter Years of Service:26

You may retire!

uoas050@elinux:~> sh retirementstatus

Enter Age: 15

Enter Years of Service:20

Your age is less than your Years of Service! This is

invalid.

Create the retirementstatus script and run with 50 for Age and 10 for Years of Service.

Run again with 62 for age and 26 for Years of Service.

Run again with 15 for age and 20 for Years of Service.

read –p “Enter Age: “ age

read –p “Enter Years of Service:” years

if [[ $age –ge $years ]]

then

 if [[ $age –ge 60 && $years –ge 10 ]]

 then

 echo “You may retire!”

 else

 if [[ $years –ge 25 ]]

 then

 echo “You may retire!”

 else

 echo “In order to retire, you must be at least age 60”

 echo “with 10 Years of Service or “

 echo “have 25 years of service.”

 fi

 fi

else

 echo “Your age is less than your Years of Service! This is invalid.”

fi

**Pattern Matching and the Case Statement**

Case statements are used to help improve readability. If you are accepting a value from a user and expect one of a certain number of responses (3 or more), then the case statement might be the best choice.

The general logic is as follows:

Case test-string

Case 1: Perform these statements if the Case1 is equal to the contents of the test-string.

Case 2: Perform these statements if the Case2 is equal to the contents of the test-string.

Case 3: Perform these statements if the Case3 is equal to the contents of the test-string.

…

Case Else: Perform these statements if none of the previous Cases are equal to the contents of the test-string.

Syntax of a Case Statement:

case test-string in

 pattern1) statements

 statements

 ; ;

 pattern2) statements

 statements

 ; ;

 patternN) statements

 statements

 ; ;

esac

read –p “Enter color: “ Color

case $Color in

red) echo “The color is red”

;;

\*) echo “the color is not red – it is “ $Color

esac

save the script as colorcase. Run it twice – once with the color blue and once with the color red as the response. The output is below.

uoas050@elinux:~> sh colorcase

Enter color: blue

the color is not red - it is blue

uoas050@elinux:~> sh colorcase

Enter color: red

The color is red

For a pattern, you can use characters or numbers. You can also use the pattern-matching symbols of \* for matching all characters, ? for matching a single character, or […] for matching a range of characters can be used.

Suppose you want to allow red, Red, or RED to be valid input in response to the question to enter a color. To do so, you would need to use the vertical bar | to separate the patterns that are possible.

Copy the file and change it to accept red, Red, and RED as the color red.

read –p “Enter color: “ Color

case $Color in

red|Red|RED) echo “The color is red”

;;

\*) echo “the color is not red – it is “ $Color

esac

Save the script as colorcase2. Run it four times with the following responses: Red, RED, red, and blue. The output is shown below.

uoas050@elinux:~> sh colorcase2

Enter color: Red

The color is red

uoas050@elinux:~> sh colorcase2

Enter color: RED

The color is red

uoas050@elinux:~> sh colorcase2

Enter color: red

The color is red

uoas050@elinux:~> sh colorcase2

Enter color: blue

the color is not red - it is blue

**Linux Scripts Looping**

**Looping with the While Statement:**

The while statement can control the number of times a set of statements is executed. The logic used is while a condition is true, then perform activities above the done statement. Once the condition is no longer true, then transfer control to the statement following the done statement.

#The following code assigns a random number to a variable called temp and

#then performs a loop up to 32767 times or until another random number is

#equal to the original random number stored in temp.

**Note—the RANDOM function uses a default seed value of 1—produces a pseudo set of random numbers that are always the same with a seed of 1. Use the format, RANDOM(seed) to get other streams of random numbers. For example, try RANDOM(3333).**

#What do you think the ((i++)) statement does?

#The break command in the if statement kills the program run.

((temp = $RANDOM))

((i = 0))

**while [[ $i -lt 32767 ]]**

**do**

 ((rn = $RANDOM))

 ((i++))

 echo 'i= ' $i ' The random number is: ' $rn

 if [[ $temp = $rn ]]

 then

 echo 'The original random number was: ' $temp

 break

 fi

**done**

 echo 'The original random number was: ' $temp

Save the script as loop1. Run the script three times.

What is the value of i and temp each time?

**Example 2:**

Instead of limiting the number of times the loop is executed, the following code stops only when the first random number (assigned to the variable temp) is repeated.

((i = 0))

((rn = 0))

 ((temp = $RANDOM))

**while [[ $temp -ne $rn ]]**

**do**

 ((rn = $RANDOM))

 ((i++))

 echo 'i= ' $i ' The random number is: ' $rn

**done**

 echo 'The original random number was: ' $temp

Save the script as loop2.

Run the script three times. Record the value of i and temp when the script stops running.