**SCRIPTING LAB GUIDE**

## 

This document serves a guide to practice what has been taught by the instructor in the training session. The course participants can use this a helper guide to practice the below, during the **student lab session** of the training. When the student practices the below material, it acts as a re-enforcer of the concepts learnt during the training.

1. **Terminology:**
2. # : Implies Root user. ( Get into the root user's environment by typing 'su -' )
   1. $ : Implies normal user . ( Exit from the root shell by typing 'exit' )

$ **cd /home**  (Go to the home directory) :

The Bold text is what the user has to type on the command line terminal. The contents inside ( ) describes the command why its used and should **not** be typed.

## **Module 1: Creating your First script**

Lab : Learning Scripting basics

Scenario:

You are new to the world of scripting and want to learn the basics quickly. The following shall help you.

**Exercise: Shell basics**:

Task 1: Create variables :

$ longname="Navin Kumar" (Variable longname is created)

$ LONGNAME="Pravin Kumar"

$ echo $longname ( Display the content of variable longname)

$ echo $LONGNAME

$ echo $longname $LONGNAME

Task 2: Using printf:

$ printf "Hello world\n" (Prints hello world and a new line)

printf "%s, %s\n" hello world (Prints the string as its provided)

$ printf "%s, %b\n" "1\t2" "1\t2" (Prints the string with interpretaion)

Task 3: Using strong & weak quotes:

$ a="a b" (Assigns the value to a)

$echo $a (Will only a b. Doesn't interpret spaces)

$echo "$a" (Prints as its provided)

$echo '$a' (Prints $a . No expansion of the variable)

Task 4: Using arguments in scripts:

$ ./args-01-show-count 1 2 3 4 'a b c' 5 6 (Displays the number of arguments)

$ ./args-03a-process-arguments.sh 1 2 3 4 'a b c' 5 6 (While loop arguments)

$ ./args-03a-process-arguments.sh 1 2 3 4 'a b c' 5 6 (For loop arguments)

**Module 2: Working with Text**

Lab : Processing  **Text Files**

Task 5: Escaping or Quoting:

$ a=hello

$ echo $a (Prints value of a)

$ echo \$a (Prints $a)

$ b=world

$ echo $a $b (Prints the interpreted value of a & b)

$ echo '$a $b' (Prints just $a $b)

$ echo 'string 1'\'' string 2' (No interpretation is done when single quotes is used)

Task 6: Using Re-direction:

$ touch hello.txt (Creates the file hello.txt)

$ ls -l (Lists all the files in the current working directory)

$ echo hello (Prints the string 'hello' to the console)

$ echo hello > hello.txt (Writes the string 'hello' to the file)

$ cat hello.txt (Displays the contents of the file hello.txt)

Task 7: Using Here document:

$ cat << EOF (Prints a multi-line string expands variables)

> Line 1

> Line 2

> My home directory is: $HOME

> EOF

$ cat << EOF (No expansion of variables is done)

> Line 1

> Line 2

> My home directory is: $HOME

> EOF

Task 8: Types of commands

$ ./searching-01-describe-commands.sh (Lists the various of types cmds in shell)

Task 9: Prints Filesystem path of the commands

$ type -p who (Prints the filesystem path of the cmd)

**Module 3: Transforming Input**

Lab : Regular Expressions capabilities in bash

Scenario:

Having obtained the basic text processing skills you are now ready to do more work using regular expressions in bash. The following shall help you.

**Exercise : Doing useful work with text files using regular expressions.**

Task 10: Basic Regular Expressions:

$ echo 'Hi how r u?' | grep --color 'how' (Matches how)

$ echo 'Hi How r u?' | grep --color 'how' ( Regex is case sensitive. Doesn't match how)

$ echo tolstoy says i am toldtoy | grep --color 'tol.toy' (Matches both tolstoy & toldtoy)

$ echo tolstoy says i am tol.toy | grep --color 'tol\.toy' (Matches both tol.toy only. Escaping . Means . Looses its special meaning and its treated as a literal character).

$ echo 123 eafg | grep --color '[aeiou]' (Matches ea. [ ] represents the set of characters )

$ echo 123 eafg | sed 's/[aeiou]/X/' (Output is: 123 Xafg. Substitutes the 1st occurrence of set mentioned in [ ] with X. )

$ echo you 123 | grep --color [^aeiou] ( ^ Complementary matching . Ie except the set of charcaters mentioned inside [ ]. The space also matches, But you can't see it.)

$ echo you 123 | grep --color '[[:digit:]]' ( Matches only the digits )

$ echo you 123 | grep --color '[^[:digit:]]' (Matches except the digits)

$ echo you 123 | grep --color '[[:alpha:]]' ( Matches the alpha characters )

$ echo you 123 | grep --color '[^[:alpha:]]' ( Matches except the alpha characters )

$ echo 1a3 1ab3 1aax3 14 | grep --color '1[[:alpha:]]3' ( Matches only 1a3 )

$ echo 1a3 1ab3 1aax3 14 13 | grep --color '1[^[:alpha:]]\*3' ( Matches the string

14 13 )

$ echo 1a3 1ab3 1aax3 18 1 | grep --color '1[^[:alpha:]]\*3' # No matches

Task 11: Regex using Interval Repitions

$ echo 123 you 123 | grep --color '[[:alpha:]]\{2\}' (Atleast 2 alpha characters in succession should match)

$ echo 123 you 123 | grep --color '[^[:alpha:]]\{2\}' (Atleast 2 matches in succession necessary. Here 4 Matches are present.

They are 12 3space space1 23)

$ echo 123 you 123 | grep --color '[^[:alpha:]]\{4\}' (No matches)

$ echo 123 you 123 | grep --color '[^[:alpha:][:space:]]\{2\}'

( It means we are looking for a match that is NOT alpha or space. Atleast 2 in succession should match . Matches 12 12 )

Task 12: Using Anchoring Matches:

( Match any number of digits but at the starting of the line)

$ echo 123 you 123 | grep --colour '^[[:digit:]]\*'

( Match any number of non-digits but at the start of the line

$ echo you 123 you | grep --color '^[^[:digit:]]\*'

( you & space matches. But we can't see the space)

( Match zero or more non-alpha but at the end of the line )

$ echo 123 you 123 | grep --color '[^[:alpha:]]\*$' (Matches space123 at the end)

Task 13: Using Back-Reference:

( Match a followed by b or c, followed by 1 or more digits , followed by xx, followed by whatever matched the set of digits, followed by x, and followed by whatever matched the afollowed by b or c )

$ echo pre ab123xx123xab post | grep --color '\(a[bc]\)\([0-9][0-9]\*\)xx\2x\1'

$ echo pre ab123xx321xab post | grep --color '\(a[bc]\)\([0-9][0-9]\*\)xx\2x\1' ( Doesn't match anything because 123 not equal to 321)

(Doesnt match anything because ab not equal to ac)

$ echo pre ab123xx123xac post | grep --color '\(a[bc]\)\([0-9][0-9]\*\)xx\2x\1'

Task 14: Using Longest Match:

(Match zero or more of the set of characters inside [ ] )

$ echo 123 eafg youyou | grep --color '[aeiouy]\*'( Matches ea youyou )

(Match zero or more of the set of characters inside [ ] and replace it with X. )

$ echo 123 eafg youyou | sed 's/[aeiouy]\*/X/' ( Outputs: X123 eafg youyou

because at the beginning of the null there is a null charcater before 123.

It matched hence sed replaced the null character with X )

(Match one or more of the set of characters inside [ ] and replace it with X. )

$ echo 123 eafg youyou | sed 's/[aeiouy]\*/X/' ( Outputs: 123 Xfg youyou )

(Even though youyou is a longer match but its not the leftmost longest match.)

(g flag in sed tells it to replace all the matches in the text)

$ echo 123 eafg youyou | sed 's/[aeiouy][aeiouy]\*/X/g' (Ouputs: 123 Xfg X)

**Module 4: Using External Tools**

Lab : Integrating External Utilities in bash

Scenario:

Having obtained the basic text processing skills you are now ready to do more work using

integrate bash with external utilities like sed, awk. The following shall help you.

**Exercise : Using external tools**

Task 15: Using sed

$ ./sed-01-user-list.sh (Prints only the users having home directory)

$ ./sed-02-family-users.sh (Prints only the users having no login)

$ ./sed-03-head.sh (Implements head functionality using sed)

Task 16 : Using awk to extract fields

$ awk -F : '$NF=/\/bin\/bash/' passwd (Prints only the lines having bin/bash as the shell. There is no action here. Print is the default action)

$awk -F : '$NF=/\/bin\/bash/ { print $3 }' passwd (Prints only the 3rd field)

$ awk -F : '($4 ~ /Pravin Kumar/) { total += $NF }

END { print "total:", total }' checkbook1.txt

(In the file checkbook1.txt find the line matching Pravin Kumar in the 4th field if so, do a total of the last field of such lines)

Task 17 : Sorting text

$ grep -v ^# checkbook1.txt | sort -t : -k6gr (Sort by amount . Neglect the comments)

$ grep -v ^# checkbook1.txt | sort -t: -f -k4,4 (Sort by recipient. Option f is to ignore case )

$grep -v ^# checkbook1.txt | sort -t: -k5r (Withdrawal comes first as Ascii value of W > D)

$ sort -t: -k1 passwd | sed 10,40d | sed 's/:.\*//' ( Sort the user names in the passwd file ; Remove 10-40 lines so that the output fits on the screen.)

$ sort -t: -k3n passwd | sed 10,40d ( Sort by user id that is the 3rd field numerically)

## **Module 5: Using Conditional Statements**

Lab : Building your scripts with conditions

Scenario:

You have come mid way in developing your scripting knowledge. Your requirements have grown and making use of conditions is a need. Also you want to know write code based on the exit status of commands. The following shall help you.

Task 18: Getting exit status of command

$ cd /tmp/x (Change to a non-existent directory)

$ echo $? (Above cmd is a failure hence returns 1 )

$ cd $HOME (Change to a existent directory)

$ echo $? (Above cmd is a success hence returns 0 )

Task 19: Using exit status with if condition

$ ./exitstatus\_if.sh

$ echo $? (Returns 0 as the last cmd is the echo cmd. It executed successfully.

the exit status of the entire if statement is that of the exit status of the last cmd run)

$./exitstatus\_not\_if.sh

$ echo $? (Returns 1)

$./exitstatus\_if\_elif.sh

$ ./exitstatus\_with\_if\_and.sh

$ echo $? (Returns zero as the exit status of the entire if statement is that of the exit status of the last cmd run. Here (echo) executed successfully.

&& operates as a shor ciruit)

$ ./exitstatus\_with\_if\_or.sh

$ echo $? (Returns zero as the exit status of the entire if statement is that of the exit status of the last cmd run. Here (echo) executed successfully.

|| operates as a shor ciruit)

(Child shell returns 42 to parent shell)

$ bash

PS1='>$'

>$ps #Shows 2 bash processes

exit 42 # Exits from the child shell. Shell prompt changes

$ echo $? #Returns 42

$ ./checkargs.sh 1 2 3 4 (if condition to check number of arguments)

$ echo $? (Returns 1)

$ ./checkargs.sh -f /some/file

$ echo $? (Returns 0)

Task 20: Using detailed if-elif-else condition

$ ./ftest.sh ftest.sh

$ ./ftest.sh /tmp

$ ./ftest.sh /dev/tty

Task 21: test cmd testing zero length

$ ./if\_zerolen.sh

Task 22: test cmd testing non empty string

$ ./if\_empty.sh

Task 23: Use case-esac

$ ./case\_esac.sh yes

## **Module 6: Using Advanced Scripting Options**

Lab : Building reusable code

Scenario:

You have come a long way in developing your scripting knowledge. Have written much scripts , now you would like to build re-usable code. So you look for Functions, working with options, sending signals The following shall help you.

**Exercise : Functions**

Task 24: Developing Functions

$ ./contrary.sh Saturday (Prints hello world or doesn't print it based on today's date)

**Exercise : Working with getopts**

Task 25: Process options with getops

$ ./getopts-demo.sh -f /x -v -l a b c (Process the options & arguments and prints only the arguments using getopts)

**Exercise : Using signals**

Task 26: Sending USR1 & HUP signals using trap

$ ./looper.sh & (Run the process in the background)

$ jobs (Prints the jobs spawned by the user)

$ kill -HUP 6415 (Send the HUP signal. The program ignores it and continues the job)

$ jobs (Can see the process still running)

$ kill -USR1 6415 (Send the SIGUSR1 signal. The program catches it and terminates it as its told so in the program )

Task 27: Interpretation of trap while defining and signal occuring

$ ./quote1.sh (Use of double quotes makes trap to remember whats the command while its defined. Hence old value of x is printed)

$ ./quote2.sh (Use of single quotes makes trap not to interpret at defining moment but when the signal occurs use the value of x at that moment. Hence new value of x is printed)