Operating System Lab

Experiment 4

Aim : Shell Scripting under Linux

Bourne Again Shell (bash)

This is the default shell of Linux. Earlier SHell was created by Steven Bourne and so the shell was named Bourne shell. This was distributed with Version 7 Unix in 1978. It was named as Bourne Again with the concept of being "born again". Bash was created in 1987 by Brian Fox and 1990 enhanced by Chet Ramey (main maintainer) and the community.

There are other shells for Linux too like C SHell (csh) created by Bill Joy, K SHell (ksh) created by David Korn and the most feature rich shell Z SHell written by Paul Falstad in 1990 when he was a student at Princeton University.

Changing Shells

Type the command name at the prompt.

```
manik@manik:~> csh        | Now I am at the default bash and changing to csh
/home/manik>              | Now I have changed to csh
/home/manik> ksh                | Now I am changing to ksh
manik@manik:/home/manik>  | Now I have changed to ksh
manik@manik:/home/manik> zsh    | Now I am changing to zsh
manik@manik:~>                  | Now I am at zshell
```

Question : How many shells have been opened?

Quitting Shells

Type ^d (equivalent to pressing exit)

Some basic information to get you started :-

Variable and their Expansion
A variable can be created by the following manner. No datatype declaration is there.

```
manik@manik:~> text="hello world"
manik@manik:~> echo "The text is :"$text
The text is :hello world
```

See that for a single word like hello, no quotes are necessary. However for this multi-word example it was essential.

```
manik@manik:~> text2=hello
manik@manik:~> echo "The text2 is :"$text2
The text2 is :hello
```

Another Example :-

```
manik@manik:~> echo "here $text2 shows :"$text2
here hello shows :hello
```

We have not desired this output. We wanted to see `here $text2 shows :hello` on screen! But in this example the variable text2 has got expanded inside the double quotes as well. So we saw that double quotes does not prevent variable expansion. We require some stricter method to prevent it.
This can be accomplished by single quotes which are very strict and don't allow inline expansion of variables.

```
manik@manik:~> echo here '$text2' shows :$text2 or
manik@manik:~> echo 'here $text2 shows :'$text2
here $text2 shows :hello
```

Pipe and backtick

As you know already, pipe sends the output of one command to another command. However we are just connecting the stdin in this case. We are not passing the output as command line arguments.
To pass the output as command line argument we can use backtick. However the direction of pipe and backtick are opposite. Pipe feeds the output of left command to right and backtick feeds the output of the right command to the left.

```
manik@manik:~/Documents/cprog> wc `ls *.c`
  8   9  77 abc.c
  7  10  76 abcd.c
  6   8  61 main.c
  6   8  58 new.c
  3  10  73 one.c
  3  11  74 two.c
 33  56 419 total
```

Finding Path

```
manik@manik:~/Documents/cprog> echo $PATH
/home/manik/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/ga
mes:/opt/kde3/bin:/usr/lib/mit/bin:/usr/lib/mit/sbin:.
```

Finding where a program is located

```
manik@manik:~/Documents/cprog> which echo
/bin/echo
manik@manik:~> whereis echo
echo: /bin/echo /usr/share/man/man3/echo.3ncurses.gz /usr/share/man/man1/echo.1.gz /usr/
share/man/man1p/echo.1p.gz
```

Continuing Lines

```
manik@manik:~/Documents/cprog>echo This \
>Is \
>  A \
>Very \
>Long \
> Command Line
This Is A Very Long Command Line
```

What is a Shell Script?
This is an executable text file with instructions. Let's create a simple one.

```
manik@manik:~/Documents/cprog> cat > hello.sh <<end
>#!/bin/bash
>echo "Hello World"
>end
manik@manik:~/Documents/cprog> chmod +x hello.sh
manik@manik:~/Documents/cprog> ./hello.sh
Hello World
```

Exit Status of a program / shell script

```
manik@manik:~> echo "Hello World"
Hello World
manik@manik:~> echo $?
0
```

Sample Program

```
#!/bin/bash
echo "Hello World"
exit 3

manik@manik:~> echo $?
3
```

A Small Test for Equality, Less than and Greater than

```
manik@manik:~> test 1 -lt 10
manik@manik:~> echo $?
0
manik@manik:~> test 1 -gt 10
manik@manik:~> echo $?
1
manik@manik:~> test 1 == 10
manik@manik:~> echo $?
1
```

Other tests

INTEGER1 -eq INTEGER2	INTEGER1 is equal to INTEGER2
INTEGER1 -ge INTEGER2	INTEGER1 is greater than or equal to INTEGER2
INTEGER1 -le INTEGER2	INTEGER1 is less than or equal to INTEGER2
INTEGER1 -ne INTEGER2	INTEGER1 is not equal to INTEGER2

Task 1
Take 3 variables a b and c with imaginary values. Compare them with each other and display results.

Control Structures1 (if)

Syntax

```
if ……..; then
…….
elif …….; then
…….
else
…….
fi
```

Shortcut if

```
[ <expression> ] && ……..
```

file : sh1.sh

```
#!/bin/bash
a=12
b=10
if test $a -eq $b ; then
echo 'a is equal to b'
elif test $a -gt $b ; then
echo 'a is greater than b'
else
echo 'a is smaller than b'
fi
exit 0
```

Run

Either make the file executable or you can invoke the file using a shell of your choice like bash

```
manik@manik:~/Documents/cprog> bash sh1.sh
a is greater than b
```

The same example done with shortcut if

file : sh2.sh

```
#!/bin/bash
a=12
b=10
[ $a -eq $b ] && echo 'a is equal to b'
[ $a -gt $b ] && echo 'a is greater than b'
[ $a -lt $b ] && echo 'a is smaller than b'
exit 0
```

AND – OR – NOT

file : sh3.sh

```
#!/bin/bash
a=12
b=10
if [ $a -lt 20 ] && [ $b -lt 20 ] ; then
echo 'a and b both are less than 20'
fi
exit 0
```

Add these lines to experiment

```
elif [ $a -lt 20 ] || [ $b -lt 20 ] ; then
echo 'any one a or b is less than 20'
elif !( [ $a -lt 20 ] && [ $b -lt 20 ] ) ;
then
echo 'a and b both are not less than 20'
```

Some other tests

-n STRING	the length of STRING is nonzero
STRING1 = STRING2	the strings are equal
STRING1 != STRING2	the strings are not equal

Task 2

Take three variables and compare them with each other and with the third variable.

Similarly experiment with strings.