Faculty of Information Technology
University of Moratuwa
Batch 21 – Level 2 Semester 1
Operating System Lab session

Operator	Description	Example	Evaluates To
+	Addition	echo $((20 + 5))	25
-	Subtraction	echo $((20 - 5))	15
/	Division	echo $((20 / 5))	4
*	Multiplication	echo $((20 * 5))	100
%	Modulus	echo $((20 % 3))	2
++	post-increment (add variable value by 1)	x=5 echo $((x++)) echo $((x++))	5 6
--	post-decrement (subtract variable value by 1)	x=5 echo $((x--))	4
**	Exponentiation	x=2 y=3 echo $((x ** y))	8

Mathematical Operators With Integers

For Mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	Meaning	Normal Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

test command or [expr] is used to see if an expression is true, and if it is true it return zero(0),otherwise returns nonzero for false.

Syntax:
test expression OR [expression]

Example:
Following script determine whether given argument number is positive.

```
if test $1 -gt 0
then
echo "$1 number is positive"
fi
test or [ expr ] works with
```

1. Integer (Number without decimal point)
2. File types
3. Character strings

Examples:
For eg. Write Script as follows
1)
$ cat > example4

```
#!/bin/sh
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]then
echo "$0 : You must give/supply one integers"
 exit 1
fi

2)
if test $1 -gt 0
then
echo "$1 number is positive"
else
echo "$1 number is negative"
fi
```

Try it as follows

```
$ chmod + example4
$ example4 5
```

Here o/p : 5 number is positive

```
$ example4 -45
```

Here o/p : -45 number is negative

```
$ example4 0
```

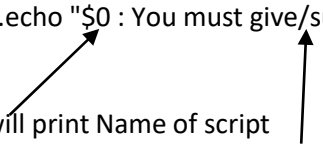
Here o/p : ./ispos_n : You must give/supply one integers

example4

Here o/p : 0 number is negative

Here first we see if no command line argument is given then it print error message as "./ispos_n :You must give/supply one integers". if statement checks whether number of argument ($#) passedto script is not equal (-eq) to 0, if we passed any argument to script then this if statement is falseand if no command line argument is given then this if statement is true. The echo command

i.e.echo "$0 : You must give/supply one integers"



1 will print Name of script

2 will print this error message

And finally statement exit 1 causes normal program termination with exit status 1 (nonzero meansscript is not successfully run), The last sample run example4 0 , gives output as "0 number isnegative", because given argument is not > 0, hence condition is false and it's taken as negativenumber. To avoid this replace second if statement with if test $1 -ge 0.

NOTE: == is equal, != is not equal.

For string Comparisons use:

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Shell also test for file and directory types

Test	Meaning
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

EXAMPLES

The following command reports on whether the first positional parameter contains a directory or a file:

```
if [ -f $1 ]
then
        echo $1 is a file
elif [ -d $1 ]
then
        echo $1 is a directory
else
        echo $1 neither file nor directory
fi
```

This example illustrates the use of test and is not intended to be an efficient method.

Logical Operators:
Logical operators are used to combine two or more condition at a time

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

if condition
if condition which is used for decision making in shell script, If given condition is true then command1 is executed.

Syntax:

```
if condition
then
command1 if condition is true or if exit status
of condition is 0 (zero)
…
…
fi
```

Condition is defined as:
“Condition is nothing but comparison between two values.”
For compression you can use test or [expr] statements or even exist status can be also used.

Loops in Shell Scripts
Bash supports:
1) for loop
2) while loop

<u>while:</u>
The syntax of the while is:

```
whiletest-commands
        do
                commands
        done
```

Execute commands as long as test-commands have an exit status of zero.

Example 01

```
#! /bin/bash
count=5
while [ $count -le 15 ]
do
        echo $count

        count=`expr $count + 1`
done
```

<u>for:</u>
The syntax of the for is:

```
for variable in list
        do
                commands
        done
```

Each white space-separated word in list is assigned to variable in turn and commands executed until list is exhausted.
Example 01

```
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

Example 02

```
#! /bin/bash
#for loops

for item in *
do
        if [ -f $item ]
```

```
        then
        echo $item
        fi
done
```

In above example you can retrieve what are the file in the current location.

The case Statement

The case statement is good alternative to multilevel if-then-else-fi statement. It enables you to match several values against one variable. It's easier to read and write.
Syntax:

```
case $variable-name in
                    pattern1) command
                    ...
                    ..
command;;
                    pattern2) command
                    ...
                    ..
command;;
                    patternN) command
                    ...
                    ..
command;;
                    *) command
                    ...
                    ..
command;;
esac
```

The $variable-name is compared against the patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is *) and it's executed if no match is found.

Examples:
if condition

1)

```
if [ -f .bash_profile ]; then
echo "You have a .bash_profile. Things are fine."
else
echo "Yikes! You have no .bash_profile!"
fi
```

2)

```
a=10
b=20
if [ $a == $b ]
then
echo "a is equal to b"
elif [ $a -gt $b ]
then
echo "a is greater than b"
elif [ $a -lt $b ]
then
echo "a is less than b"
else
echo "None of the condition met"
fi
```

for and while loop

3)

```
for i in 1 2 3 4 5
do
echo “welcome $i times ”
done
```

4)

```
for i in {1..5}
do
echo “welcome $i times ”
done
```

5)

```
for i in {0..10..2}
do
echo “welcome $i times ”
done
```

6)

```
for (( c=1;c<=5;c++))
do
echo “welcome $c times ”
done
```

7)

```
n=10
while [$n -le 20]
```

```
do
echo “welcome”
n=$((n+1))
done
```

switch case

8)

```
FRUIT="kiwi"
case "$FRUIT" in
"apple") echo "Apple pie is quite tasty."
;;
"banana") echo "I like banana nut bread."
;;
"kiwi") echo "New Zealand is famous for kiwi."
;;
esac
```