# Commands and Syntaxes for Shell Programming:

(1) echo [options] [string, variables...]

Displays text or variables value on screen.

**Command Line Arguments:**

(1) The $# Variable

Whenever you execute a shell program, the special shell variable $# gets set to the number of arguments that were typed on the command line.

(2) The $\* Variable

The special variable $\* references all the arguments passed to the program. This is often useful in programs that take an indeterminate or variable number of arguments.

(3) The $0 Variable

Whenever you execute a shell program, the shell automatically stores the name of the program inside the special variable $0. This can be used to advantage when you have two or more programs that are linked under different names and you want to know which one was executed. It’s also useful for displaying error messages because it removes the dependency of the filename from the program. If the name of the program is referenced by $0, subsequently renaming the program will not require the program to be edited

(2) **if...else...fi**

If given condition is true then command1 is executed otherwise command2 is executed.

Syntax:

if condition

then

command1 if condition is true or if exit status

of condition is 0(zero)

...

...

else

command2 if condition is false or if exit status

of condition is >0 (nonzero)

...

...

fi

(3) **Multilevel if-then-else**

Syntax:

if condition

then

condition is zero (true — 0)

execute all commands up to elif statement

elif condition1

condition1 is zero (true — 0)

execute all commands up to elif statement

elif condition2

condition2 is zero (true — 0)

execute all commands up to elif statement

else

None of the above condtion,condtion1,condtion2 are true (i.e.

all of the above nonzero or false)

execute all commands up to fi

fi

(4) **for loop**

Syntax:

for { variable name } in { list }

do

execute one for each item in the list until the list is

not finished (And repeat all statement between do and done)

done

(5) **while loop**

Syntax:

while [ condition ]

do

command1

command2

command3

..

....

done

(6) **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;;

\*) command

...

..

command;;

esac

(7) **The read Statement** Use to get input from keyboard and store them to variable.

Syntax:

read varible1, varible2,...varibleN

**Basic Operators in Shell Scripts:**

**Relational Operators**

|  |  |  |
| --- | --- | --- |
| ***Operator*** | ***Description*** | ***Example*** |
| -eq | Checks if the values of two operands are equal or not, if yes then condition becomes true. | [ a — eqb ] is not true. |
| -ne | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | [ a — neb ] is true. |
| -gt | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | [ a — gtb ] is not true. |
| -lt | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | [ a — ltb ] is true. |
| -ge | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | [ a — geb ] is not true. |
| -le | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | [ a — leb ] is true. |

**Executing a Shell Script:**

If script\_name.sh is the name of source shell program, then to execute it you have type

./script\_name.sh

or

sh script\_name.sh