Command substitution:

```
who | wc –l                    or              w|wc -l
```

```
users=`who | wc -l`
```

```
echo $users
```

Exit status:

Exit status is a numerical value returned by a command once its execution gets over. $? is the variable that is used to retrieve the exit status of the previous command that was executed. Any scripts or command line programs must terminate with a proper exit status.

The Linux shell returns an exit code for each command line program or script executed. It returns zero for a successful operation and a non-zero value ranging from 1 to 255 for a non-successful operation.

```
pwd
echo $?
cat file3; echoe "Wrong command"
echo $?
```

The exit status of the previous command is non-zero. It was unsuccessful.

Bash comparison operators:

Integer comparison operators:

Operator	Meaning	Usage
-eq	equal to	[$a -eq $b]
-ne	not equal to	[$a -ne $b]
-gt	greater than	[$a -gt $b]
-ge	greater than or equal to	[$a -ge $b]
-lt	lesser than	[$a -lt $b]
-le	lesser than or equal to	[$a -le $b]

String comparison operators:

Operator	Meaning	Usage
==	equal to	["$a" == "$b"]
!=	not equal to	["$a != "$b"]
-n	not a null string	[-n "$a"]
-z	null string	[-z "$a"]

File operators:

Like comparison of integers and strings can be evaluated, the status of any file or directory can also be evaluated in if statements.

File operators	**True if..**
-a file	file exists
-f file	file exists and is a regular file
-d file	file exists and is a directory
-r file	file is readable by the current user
-w file	file is writable by the current user
-x file	file is executable by the current user
-s file	file exists and in non-empty

test command: It is used to evaluate comparison expressions.

```
[root@machine1 /]# num=10
[root@machine1 /]# test $num -gt 10
[root@machine1 /]# echo $?
[root@machine1 /]# test $num -eq 10
[root@machine1 /]# echo $?
```

SELECTION CONSTRUCTS

The flow of the execution of script is always sequential. But if we want to change the flow of the execution based on a condition, we can make use of selection constructs.

if statement
if..else statement
Nested if statement

if statement:

Syntax:

```
if [ test expression ]
then
  block of commands
fi
```

```
#!/bin/bash
filename=/tmp/sample.txt
if [ -f $filename ]
then
  echo  “file1 is a regular file.”
fi
```

if..else statement

Syntax:

```
if [ test expression ]
then
		block of commands
else
		block of commands
fi
```

```
#!/bin/bash
filename=/tmp/sample.txt
if [ -f $filename ]
then
  echo “$filename is a regular file.”
else
  echo "$filename is not a regular file."
fi
```

if ..elif..else statement:

Syntax:

```
if [ test expression ]
then
		block of commands
elif [ test expression ]
then
		block of commands
.
.
else
		block of commands
fi
```

```
#!/bin/bash
filename=/tmp
if [ -f $filename ]
then
  echo “$filename is a regular file.”
elif [ -d $filename ]
then
  echo "$filename is a directory."
else
  echo "$filename is neither a regular file
nor a directory."
fi
```

Nested if statement:

if statement can be nested within another if statement if required.

```
#!/bin/bash
filename=/tmp/sample.txt
if [ -f $filename ]
then
 if [ -s $filename ]
 then
   echo " $filename is a non-empty regular file."
 else
   echo " $filename is an empty regular file."
 fi
else
  echo " $filename is not a regular file."
fi
```

Case statement:

Syntax:

```
case <val> in
choice1) <block of commands>;;
choice2) <bloc of commands>;;
.
.
*) <block of commands>;;
esac
```

```
#!/bin/bash
string=“chennai"
test $string == “Chennai"
case $? in
0) echo "The condition is true";;
1) echo "The condition is false";;
*) echo "Unknown exit status";;
esac
echo "Out of the case statement"
```

```
#!/bin/bash
num=100
case $num in
0) echo "The number is zero";;
10) echo "The number is ten";;
100) echo "The number is hundred";;
*) echo "No match is found"
esac
echo "Out of the case statement"
```