**UNIX** is a very popular, multiuser, multitasking, time sharing OS. Flavors of UNIX are AIX (Advanced IBM Unix), HP-UX, Linux, Solaris etc. Unix was written by Ken Thompson in ‘C’ language.

**Features of Unix: -**

1. **Portability**
2. **Security**
3. **Background Processing**
4. **Pipes**
5. **Stable and Reliable**
6. **Shells**

**Compilers**

**DB**

**Other** **Applications**

**Unix Commands**

U

**KERNEL**

**SHELL**

**SHELL SHELL**

**SHELL**

**UNIX** follows layered approach. Kernel is the important part of Unix system. It is a collection of programs which directly communicates with hardware. It is that part of system which loaded into memory when system is booted. It manages System resources, time between users and processes etc.

A program is just an executable file where as a process is an instance of a program in execution.

**SHELL: -** It is a command interpreter. User communicates directly with shell. Then shell translates user communication to functions understandable by kernel. It is programmable.

**Directories in Unix: -**

**Home: -** Home directories for users.

**Bin:** - Executable system functions like sh, rm etc.

**Dev:** - Primary location for special files or device files.

**Etc:** - System configurable files such as passwd, shadow etc.

**Tmp:** - Temporary files.

**Lib:** - Kernel related files.

**Usr:** - User related files.

**Features of UNIX file system: -** Hierarchical structure, Create and delete files, Dynamic growth of files, Protection of file data, i-node no, File permissions like Read (R), Write (W), Execute (X) etc.

**Commands:** -

**Cal** -> Calendar

**Who** -> Displays current user connected

**Date** -> System date and time

**Clear** -> Clear the screen

**More** -> Allows page wise display

**Whoami** -> Displays username/ who am I -> details about user

**Tty** -> Name of terminal

**Uname** –a -> Displays system information like Kernel name, Version, OS, Hardware name etc.

**Man** -> Displays manual

**Bc** -> Opens arbitrary calculator, to quit press EOF character (ctrl + d)

**Echo** -> Echo’s to screen what we type after echo

**Printf** -> Write formatted output.

**Type** -> Displays location of commands.

**Which** -> Looks for a file of a particular command.

Get a count of all files and directories in current directory -> **echo \* | wc**

Get a count of all files in current directory -> **echo \*.\* | wc**

Get a count of all directory in current directory -> **echo \*/ | wc**

**Ls** -> List contents of a directory -> **-r (reverse list), -a (hidden files), -I (inode numbers) –l (long format)**

**Cat** -> Concatenate and display files -> **cat a.txt (Displays content of a file), cat >> a.txt (Append to a file)**

**Cat > a.txt (Create a new file)**

**Cp** -> Copy Files -> **cp a.txt b.txt**

**Mv** -> Move/rename to a file -> **mv a.txt b.txt**

**Pwd** -> Present Working Directory

**Cd** -> Change Directory -> **cd .. (Parent Directory / back by 1 directory) “~” symbol of Home Directory.**

**Mkdir** -> Create new directory

**Rmdir** -> Remove directories (if it is empty)

**Rm** -> Remove files/directories -> **-f (forcefully), -r (recursively), -I (ask before deleting)**

**Ln** -> Link to a files. A link is nothing but a directory entry to a file.

1. Hard Link -> **ln a.txt b.txt** (should not exist already)
2. Soft Link -> **ln –s a.txt /tmp/b.txt** (should not exist already)

|  |  |
| --- | --- |
| **HARD LINK** | **SOFT LINK** |
| Same i-node numbers | Different i-node numbers |
| Original file deleted then link not delete | Original file deleted then link remains as invalid link |
| More Efficient | Less Efficient |
| Cannot used to link a file to another system | Can be used to link a file to another system |
| Cannot used to link directories | Can be used to link directories |

**Touch** -> Update access and modification time of a file/directory.

**Touch [-amc] [mmddhhmm[yy]] filename**

**Touch a.txt (Create new file)**

**“\*”** -> Matches 0 or more characters **“?”** -> Matches single character

**“[]”** -> Matches 1 single character defined inside []

When a pipe is established between two commands the output of 1st command will be redirected as standard input of 2nd command. Pipe is a mechanism through which process communicates with each other.

**Tr** -> Translate characters -> It is a filter program.

**Tr “a” “b” -> Converts any occurrence of letter “a” to letter “b”**

**Tr –s “ “ “:” -> Multiple occurrences of white spaces between the words replaces with “:”**

**Cmp** -> Compare 2 files -> **-s returns an exit code**

**Cut** -> Cut selected field of each line of a file -> **-f is a list of filed and –d is a delimiter**

**Cut –f 1,3 –d “:” a.txt - > Extract 1st and 3rd field from file.**

**Cut –c 2-5, 8-10 a.txt -> Extract 2nd to 5th and 8th to 10th characters from all lines of a file.**

**Paste** -> Merging corresponding lines of given files. It does horizontal pasting.

**Sort** -> To sort contents of a file. -> **-d (directory) / -n (numeric) / -r (reverse) / -k n (sort on field n)**

**Sort –t “:” –k 3 a.txt -> -t is a delimiter and 3 is a field**

**Head** -> Displays first count lines of a file (10 by default). If no file specifies it reads from standard command line.

**Tail** -> Displays last count lines of a file

**WC -> Displays no. of. Words (-w) / no. of. Lines (-l)/ no. of. Charcters (-c) in a file.**

**Uniq** -> To remove adjacent repeated lines.

**-u (displays only lines not repeated in file)**

**-d (displays repeated lines in file)**

**-c (precede count of line number of times it occurs)**

Regular Expression is a string of meta characters which can be used to match more than 1 type of pattern. Used in grep, egrep etc.

**Grep** -> Pattern matching in a file. Displays line containing the pattern to standard output.

**-c (Report only the no. of. Matching lines)**

**-l (list only names of files containing pattern)**

**-v (displays all lines expect those containing pattern)**

**Anchor -> “^” (beginning of line), “$” (end of line), “\<” (beginning of word), “/>” (end of word)**

**Egrep ‘you|You’ a.txt (matches lines containing you or You)**

**Hel (l|llo) matches either Hell or Hello**

**Ab \ {2,4\} matches abb, abbb, abbbb -> 2 represent atleast position and 4 represent atmost position**

**Ab \ {2\} matches abb exactly 2 b’s.**

**A process is a program in execution. Each process is allocated a process Id called as PID.**

**Ps -> Give details of a process**

**-f gives full listing**

**-e / -a displays all process**

**Nice -> Execute a command with lower priority. Default is 10. Nice to other processes.**

**Kill -> Send signal to process.**

**-9 Terminate Process**

**-15 Software Termination Signal (Default)**

**Time -> Sum of User time and System time represents CPU time.**

**Jobs -> List the status of all jobs -> -1 displays process id.**

**FTP** protocol establishes 2 connections with host. One of the connection is used for data transfer and second one is used for control purpose. It remains active throughout session while data transfer connection is open and closed for every file transfer. ASCII and Binary are modes of file transfer in FTP.

**SSH** (Secure Shell Conn) it is secure and encrypted form of remote connection.

**SCP** uses **ssh** connection for file transfer.

The 2 imp files are used frequently for password authentication **are /etc/passwd (normal user view) & /etc/shadow (admin view)**

**Passwd** -> Command used to change user password.

**Class of users for a file -> Owner/ Group/ Others**

**2 modes to set permission to a file -> Absolute (Octal) / Symbolic (String)**

**Owner -> Read (4), Write (2), Execute (1)**

**Group -> Read (4), Write (2), Execute (1)**

**Others -> Read (4), Write (2), Execute (1)**

**“-“ indicates no permission set**

**(rwxrw-r--) -> 1st set of rwx stand for owner -> 2nd set of rw- for Group -> 3rd set of r—for Others)**

**Default permission for a file rw-rw-rw- (666) and for directory rwxrwxrwx (777)**

**U + rx -> indicates that user has read and execute access.**

**Chmod -> Used for setting file and directory permission.**

**Chown -> Change ownership of a file.**

**Chmod 777 a.txt #absoulte mode**

**Chmod ugo + rwx a.txt #string mode (Symbolic Mode)**

**“r” -> Only list contents of directory nothing else is possible.**

**“w” -> Nothing can be done.**

**“x” -> Can enter in directory only but nothing else is possible.**

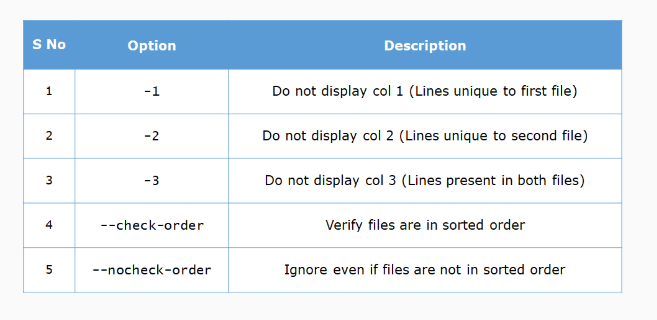
**“w” and “x” -> Everything is possible except listening of file.**

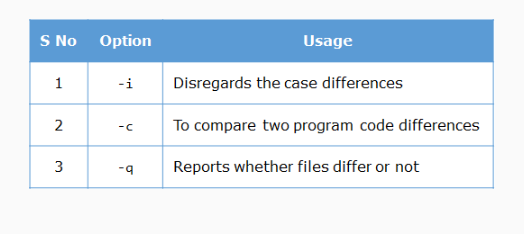
**“r” and “x” -> Can’t create a new file but edit/append/delete existing ones.**

**“r” and “w” -> Nothing is possible because can’t enter into directory**.

**More Information about UNIX:** -

**Comm** commands compares two sorted files line by line and produces three column output when no options were passed to it. Column one contains lines unique to file1, column two contains lines unique to file2, and column three contains lines common to both files.

The command "**diff**" compare files line by line. This command becomes helpful when you wish to know the exact record level differences. It tries to determine the smallest set of deletions and insertions between files. Typically, diff is used to show the changes between two versions of the same file.



diff  Brands\_old.dat Brands\_new.dat

1,2c1,2

< samsung

< samsung

---

>

> sony

Let us understand the above output. The first line of the output contains the following

* Line numbers from the first file
* A letter (**a** for add **d** for delete **c** for change)
* Line numbers from the second file

So, 1,2c1 means that the lines from 1,2 in first file need to be changed to match the line number 1 in second file.

The second and third lines in the above output are the lines from first file followed by a separator and then the next line is from second file.

**Note:**

* Lines preceded by a "<" are lines from the first file
* Lines preceded by ">" are lines from the second file

diff -c Brands\_old.dat Brands\_new.dat

The option "c" is used with "diff" command to produce the following result.

\*\*\* 1,5 \*\*\*\*

! samsung

! samsung

sony

sony

sony

--- 1,5 ----

!

! sony

sony

sony

sony

* ! symbol prefixed to few lines where the change is expected
* + symbol prefixed to lines indicates that this line in second file, needs to be added to the first file
* -  symbol Indicates a line in the first file that needs to be deleted

**SORT**

10.123.80.65 - Admin [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 200 2048

10.123.80.65 - Admin [11:10:15] "GET /samsung-Inverter-sam98009/index.html HTTP/1.0" 206

10.123.80.65 - User1 [11:10:15] "GET /sony-Mobile-Mob8009/index.html HTTP/1.0" 200 1024

10.123.80.65 - User2 [11:10:15] "GET /samsung-Mobile-sam88009/index.html HTTP/1.0" 302

10.123.80.65 - User1 [11:10:15] "GET /sony-Mobile-Mob8009/index.html HTTP/1.0" 304

10.123.80.65 - User2 [11:10:15] "GET /sony-Laptop-Lap8009/index.html HTTP/1.0" 403

10.123.80.65 - User3 [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 304

10.123.80.65 - User4 [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 404

10.123.80.65 - User5 [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 200 512

10.123.80.65 - User5 [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 200 4096

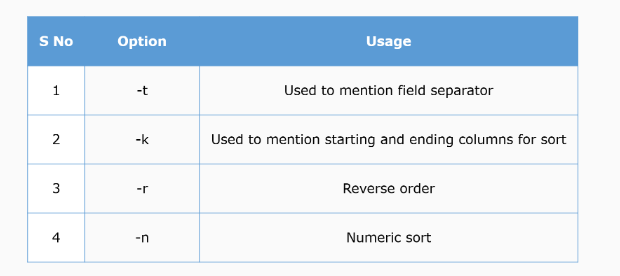
10.123.80.65 - User4 [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 403

10.123.80.65 - Admin [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 304

12.123.80.60 - Admin [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 200 2048

12.123.80.60 - Admin [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0" 200 1024

12.123.80.60 - Admin [11:10:15] "GET /sony-laptop-Lap789/index.html HTTP/1.0" 200 1024



With the help of "**sort**" command, a report the number of users searching for products ordered by the number of times they appear can be found as shown below.

cut -d ' ' -f1,3 apache\_access.log |uniq -c|sort –r

3 12.123.80.60 Admin

2 10.123.80.65 User5

2 10.123.80.65 Admin

1 10.123.80.65 User4

1 10.123.80.65 User4

1 10.123.80.65 User3

1 10.123.80.65 User2

1 10.123.80.65 User2

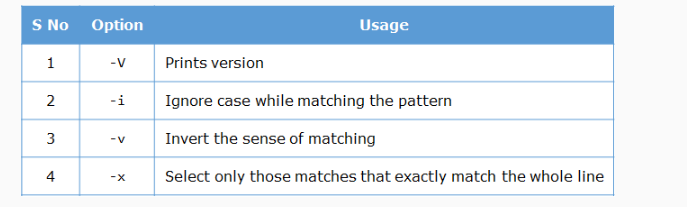
1 10.123.80.65 User1

1 10.123.80.65 User1

1 10.123.80.65 Admin

**GREP:** -

The content of a file or text input is processed line by line, to search for the pattern and the lines matching the pattern are displayed on the terminal. It can be used to search for lines of text that match one or many regular expressions.



If you need to find a pattern from the beginning of every line, execute the following command

grep '^12' access.log

12.123.80.60 - Admin [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0"

12.123.80.60 - Admin [11:10:15] "GET /sony-Inverter-IN8009/index.html HTTP/1.0"

12.123.80.60 - Admin [11:10:15] "GET /sony-laptop-Lap789/index.html HTTP/1.0"

**The command "grep" substitutes meta characters as strings and does not interpret their special meaning unless they are escaped with a '\’. To interpret the special meaning of meta characters the command "grep" with option "-E" is to be used.**

The command "**umask**" is used for computing the file permissions for a newly created file. A mask is a group of bits which restrict how the corresponding permission bit is set for a newly created file.

The above command shows the default value of the umask set in the Unix environment. Alternatively, you may also use the following command to get to know the permissions of newly created files and directories.

umask –S

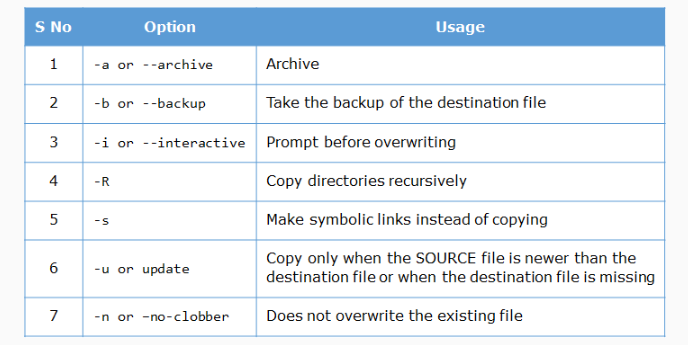
u=rwx,g=rwx,o=rx

To set a new value to umask, you can use the following command.

umask 0022

cp command: -

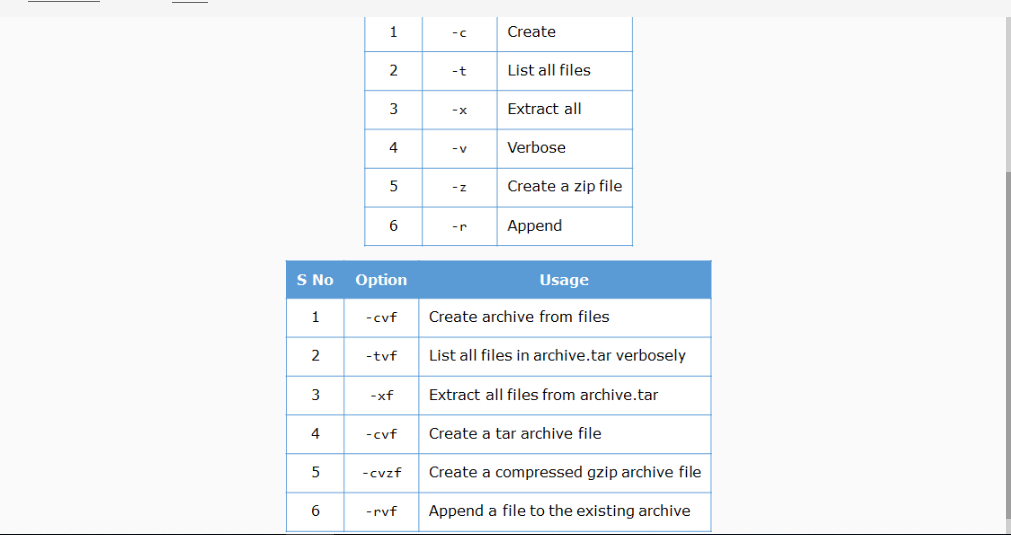
"**cp**" command is used to copy a file from one directory/partition to another, possibly on a different file system. The original file remains unchanged, and the new file may have the same or a different name. Usually, the new file is placed in another partition to use it as backup of the original file.



# Archiving files: -

Old data from the application may need archival, as it may not be in frequent use. Achieving compresses the data and hence reduces the storage space required.

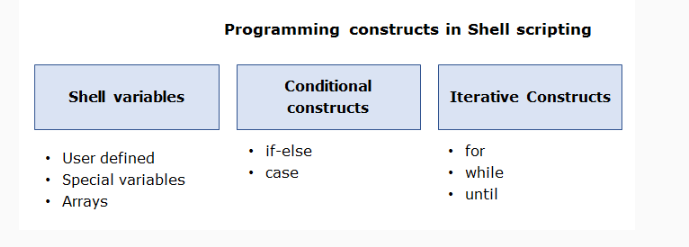
‘**tar’** command saves many files together into a single tape or disk archive, and can restore individual files from the archive options.



To create an archive file, use the "tar" command with the options mentioned in the following example

tar -cvf application\_files.tar /home/er111/Unix\_Course/backup\_directory/

Shell Scripting: -



# Shell variables: -

As in any programming language, to handle data, variables are used .You need to learn about three types of shell variables namely

* User defined variables
* Arrays
* Special variables

**Shell variables are identifiers that take a value at command prompt. Shell variables can be accessed using a '$' in front of the variable name. The name of a variable can contain only letters (upper or lower case), numbers (0 through 9) or the underscore character. Quoted variables preserve white spaces where as non-quoted variables don’t. Non quoted variables will split the variable in parts at white spaces. An argument enclosed in double quotes presents itself as a single word.**

**Sample shell script: -**

1. *#!/usr/bin/bash*
2. var1=10
3. var2=20
4. echo $var1
5. echo $var2

The first line of the shell script has the full path of the interpreter which will execute this file. There after two variables var1 and var2 are created and echoed to the terminal.

**Executing the shell script**

Let’s first understand how to execute a shell script

1. Execute shell script by specifying the interpreter in which the shell interpreter is mentioned in the command line

**sh <file-name.sh>**

2. Execute shell script using file name and it is executed by the current shell without forking a sub shell

**./<file-name.sh>**

The following shell script takes input from the user and prints the appropriate output. The following code explains use of "**readonly**" key word to create constants.

**echo -n "Please enter your name: " # Observe the -n option keeps the cursor on the same line #**

**read NAME**

**# what if I do not want to change this variable anywhere inside the script? #**

**# Simple, make it behave as a constant or read-only #**

**# Can be done as below #**

**readonly NAME**

**# The below operation is not permitted on a read-only variable #**

**NAME="New Name"**

# Arrays in shell: -

Arrays are the collection of elements. In shell scripting, data type declaration is not required with arrays. The values of the variables can be accessed using '$' as prefix to these elements. Arrays provide a method to group a set of data. Create and initialize array in shell script in the following way.

ARRAY=(10 20 30 40 50 60)

ARRAY\_2=(10 20 30 “Infosys” “Amrit”)

#!/usr/bin/bash

# Arrays in BASH Shell

# These are Shell Specific, since bash is very popular, it is considered here

ARRAY=(10 20 30 40 50 60 70 "Infosys" "Limited")

# To print the first element

echo -n "The first element is: "

echo $ARRAY

# OR

echo -n "The first element is: "

echo ${ARRAY}

# To display all the elements at one time #

echo -n "The array is (in \* form): "

echo ${ARRAY[\*]}

# OR

echo -n "The array is (in @ form): "

echo ${ARRAY[@]}

# Find the length of an array

echo "Length of the array is "${#ARRAY[\*]}

# Find a particular element

echo "Second Element is ${ARRAY[1]}"

# Find the length of an element

echo "Length of Second Element is: "${#ARRAY[1]} # Observe no concatenation operator

The first element is: 10

The first element is: 10

The array is (in \* form): 10 20 30 40 50 60 70 Infosys Limited

The array is (in @ form): 10 20 30 40 50 60 70 Infosys Limited

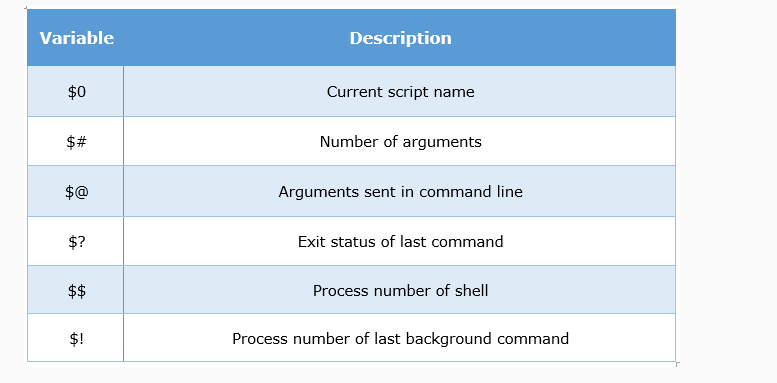
Length of the array is 9

Second Element is 20

Length of Second Element is: 2

# Special variables: -

Unix system has few variables with special purpose. The following table depicts frequently used special variable.



#!/bin/bash

#This file accepts three parameters and displays them on the terminal

echo "\n The first argument received is $1, the second $2 and the third is $3\n"

#!/bin/bash

#This file accepts three parameters and displays them on the terminal

echo "$# arguments were passed to the script. They are $@"

exit

echo "\n The first argument received is $1, the second $2 and the third is $3\n"

shift

echo "Shifting"

echo "\n The first argument received is $1, the second $2 and the third is $3\n"

./positional\_parameters.sh First Second Third

3 arguments were passed to the script. They are First Second Third

1. **The special variable $# gives the number of arguments passed to the script at command line**
2. **The special variable $@ preserves the values of the arguments passed at command line**
3. **The command "exit" takes the control out of the shell script without executing the next set of statements**

**The following demo shows the use of command substitution in Unix shell scripting.**

#!/usr/bin/bash

1 # This example demonstrates the Command Substitution in Unix #

2 **VAR=`ls -l`**

3 # The following output will be a white space separated string #

4 #**echo $VAR**

5 # However, when you do the following, you can observe newline being added #

6 **echo "$VAR"**

**# Using multiple commands in conjunction #**

**VAR=`echo "Infosys Limited";ls -l`**

**echo $VAR**

**VAR=`echo "Infosys Limited" | wc -w`**

**echo $VAR**

**# Note: The above form has been deprecated and now re-written as follows #**

**VAR=$(echo "Infosys Limited" | wc -w)**

**echo $VAR**

# Variable substitution: -

# 

**Input output statements in shell**

Shell scripting allows user to provide input either through command line or in an interactive way when probed. The command "**read**" is used to take the input from the user.

**Syntax:** **read  [-p prompt] [-a array] [-d delim] [-ers] [-n nchars]**

# echo " Multiplication of two numbers in shell script"

# read -p "Enter a number greater than zero: " var1

# read -p "Enter another number greater than zero: " var2

# read -p "Press any key to see the answer"

# echo "The result is: "

# let result=$var1\*$var2

# echo $result

**With the inclusion of the option "-n", followed by 1, after encountering one character, control moves to the next statements. However, the character entered by the user is echoed on to the screen. This can be avoided by using the option "-s".**

echo " Multiplication of two numbers in shell script"

read -p "Enter a number greater than zero: " var1

read -p "Enter another number greater than zero: " var2

**read -sn1 -p "Press any key to see the answer :"**

echo

echo "The result is: "

let result=$var1\*$var2

echo $result

**Shell script supports conditional constructs such as if -else and case statements. The following is the syntax to be used for if-else construct.**

**if [ condition ];**

**then**

**<statements>**

**else**

**<statements>**

**Fi**

**If-else**

**if [ condition ];**

**then**

**<statements>**

**elif [ condition ]**

**<statements>**

**else**

**<statements>**

**Fi**

**Case**

**case variable in**

**option 1)**

**statements to be executed if option 1 matches**

**;;**

**Option 2)**

**statements to be executed if option 2 matches**

**;;**

**\*)**

**default statement**

**esac**

#!/bin/bash

if [[ $# -eq 0 ]] ; then

echo -e " Please enter the arguments at command line!!!\n"

echo -e " You have entered $(basename $0) <arg1> ...<argn>\n"

exit 1

else

echo -e " You typed the following arguments: $@\n"

fi

**Output: -**

Please enter the arguments at command line!!!

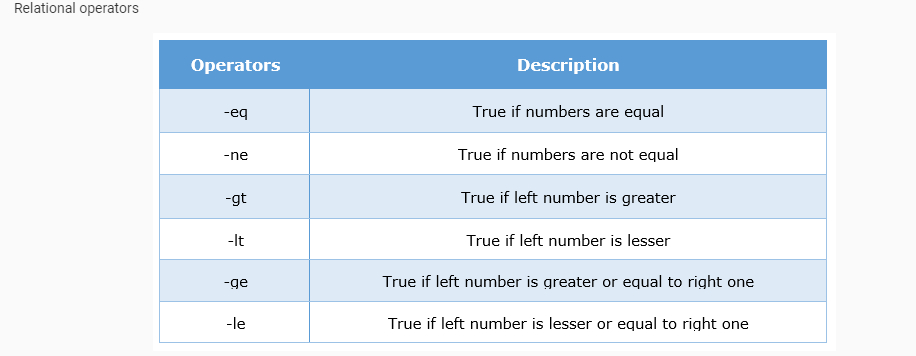
You have entered if.sh <arg1> ...<argn>

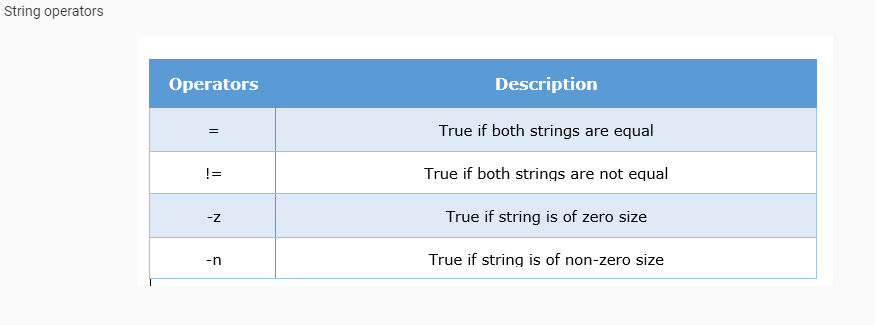
**Operators in shell: -**

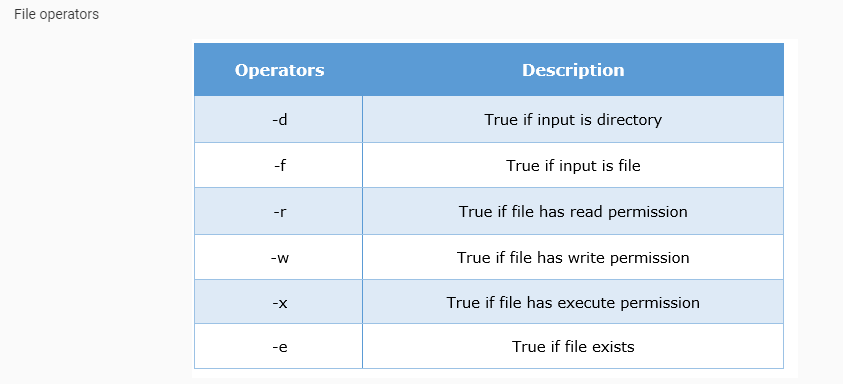
Shell supports the following categories of operators

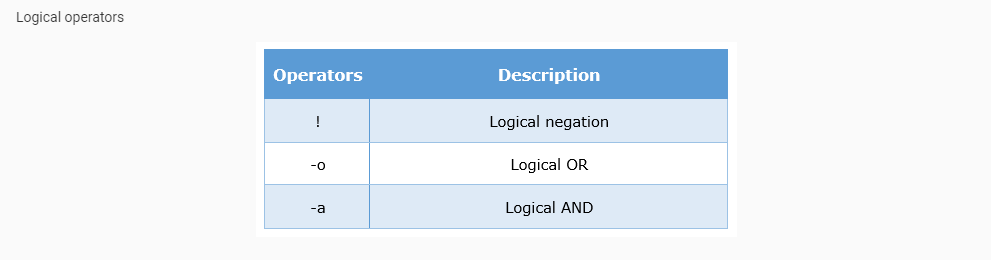
* **Arithmetic operators**
* **Relational operators**
* **String operators**
* **File operators**
* **Logical operators**

All arithmetic operators are supported as in any other programming language. The following tables shows the relational, string, file and Boolean operators.









Shell scripting supports the use of looping constructs just as in programming languages. The following are the iterative constructs in shell.

* **for loop**
* **while loop**
* **until loop**

for variable in <list>; do

<statements>

Done

while [ condition ] ; do

<statements>

Done

until [ condition ] ; do

<statements>

Done

**Difference between while and until loops**

**The while looping construct executes the statements when the given condition evaluates to true. On the other hand, the "until" loop executes the statements as long as the condition evaluates to false.**

#!/usr/bin/csh

# For Loop #

ARR=(10 20 30 a b c 50 60)

for VAR in ${ARR[\*]}

do

echo "Value now is: $VAR"

done