Unit IV Introduction to Unix and Shell Programming

Following major topics will be covered in this unit

- Unix file system
- Unix Shell
- Unix Commands
- Shell variables
- Basic concepts of Shell programming

Introduction to Operating System

- An operating system (OS) is a resource manager
- An OS is a program that manages computer hardware
- It controls the entire operations of the computer system
- It takes the form of a set of software routines that allow users and application programs
- to access system resources
- Ex: the CPU, memory, disks, modems, printers network cards etc.)
- in a safe, efficient and abstract way etc.

Introduction to Unix

- UNIX was earlier known to be UNICS
- which stands for UNiplexed Information Computing System..
- UNIX is a popular operating system, first released in 1969
- UNIX is a multi-tasking, powerful, multi-user, a virtual OS
- which could be implemented on a variety of platforms

History of Unix OS

- The development of UNIX system got started at Bell Labs by mainly Ken Thompson, Dennis Ritchie
- The first version of this operating system was written in assembly language
- But later during 1973, Version 4 was written in C language
- During the 1990s, the UNIX system started gaining popularity as the Linux distributions developed
- In 2000, Apple released its own UNIX system, known as Darwin, which later became MacOS.

Components of Unix OS

- The UNIX operating system basically comprises three components:
- Shell
- Program
- Kernel
- Shell: It acts as an interface between the kernel and the user
- The user has to go under the authentication check before entering into the shell
- Program: It can be said that everything inside UNIX is either a file or a program

- A process is a program under execution having unique PID(program Identifier), used to identify it
- Kernel: The Kernel is responsible for allocating time and memory to programs
- It also handles file storage and communication while responding to the system calls.

Characteristics of Unix

- Portability: This feature helps the user to change and compile the code on a new machine
- Multi-tasking: more than one process to run at a time if one is running, other can also run background
- Pipes and Filters: which help in making of complex programs from simpler programs
- Shell: The shell hides the intricate hardware details from the user
- Extensive Library: has a support of extensive library which makes it a useful and relevant

Advantages of Unix

- The main advantage of Unix is its portability which helps the user to run program on different machine
- UNIX makes minimum use of physical memory usage while running the various tasks flawlessly
- UNIX is capable to perform complex tasks with the minimal load and efficiently
- It supports Hierarchical File system which helps for easy maintenance and efficiency
- UNIX system is secure due to its strong server validation and authentication

Disadvantages of Unix

- UNIX operating system is command line based which increases the difficulty for the casual users to use it
- It is meant for expert programmers who know command line commands very well
- Documentation of various UNIX tools is hard to find
- The commands used are cryptic, makes use of special characters which makes it difficult for new users

Unix File System

The File

- The file is a container for storing information
- All file attributes are kept in a separate area of the hard disk
- Not directly accessible to human, but only to kernel
- Unix treats directories and devices as file as well
- All physical devices like hard disk, memory, CD-ROM, printer and modem are treated as files
- The shell is also a file, and so is the kernel
- Unix treats the main memory in the system, it’s a file too

Files Category

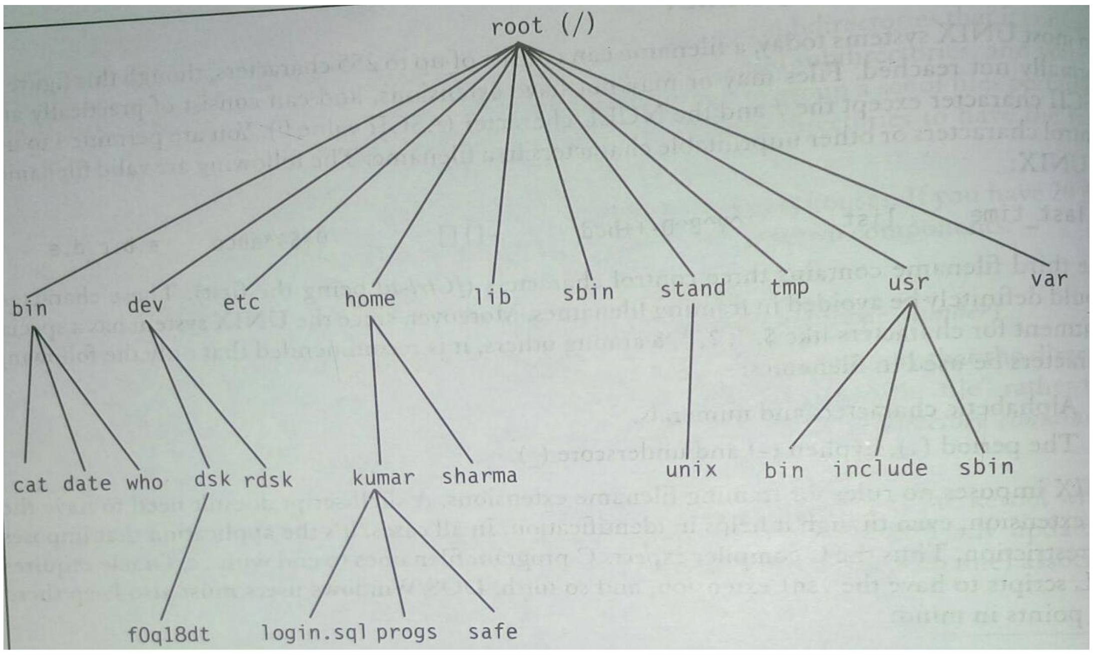
- Ordinary file: Also known as regular file
- It contains only data as a stream of characters
- Directory file: A directory contains files and other directories

- It contains their names and a number associated with
- Device file: All devices and peripherals are represented by files

Unix File System

- All files in Unix are related to one another
- The file system in Unix is a collection of all of these related files
- These related files are organized in a hierarchical (an inverted tree) structure
- The implicit feature of every Unix file system is that there is a top
- This top is called root and is represented by /(slash)
- Root is actually a directory

The Unix File System Tree



Description of Unix File System

- The root directory (/) has a number of subdirectories under it
- These subdirectories in turn have more subdirectories and other files under them
- Every file, apart from root, must have a parent, and it should be possible to trace a file to root
- In these parent – child relationships, the parent is always a directory
- When we log on to the system, Unix automatically places us in a directory called the home directory
- Home directory is created by the system when a user account is opened
- The shell variable HOME knows home directory
- $echo $HOME
- /home/kumar First / represents the root directory
- The home directory is determined by the system administrator at the time of opening a user account
- Its pathname is stored in the file /etc/passwd
- On many Unix systems, home directories are maintained under /home

Unix files related Commands

- pwd: Checking for current directory
- cd: Change the current director
- mkdir: Making directories
- rmdir: Removing directories
- ls: Lsiting directory contents

Unix File System

- /bin and /usr/bin – these are directories where all the commonly used Unix commands are found
- /etc – this directory contains the configuration files of the system
- /dev – this directory contains all device files
- /lib and /usr/lib – contains all library files in binary
- /usr/include – contains standard header files used by C programs
- /tmp – users are allowed to create temporary files
- /var – the variable part of the file system

Unix Shell & Variables

Unix Shell:

A Unix shell is a command-line interpreter or shell that provides a command line user interface for Unix-like operating systems. The shell is both an interactive command language and a scripting language, and is used by the operating system to control the execution of the system using shell scripts.

A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

Shell is an environment in which we can run our commands, programs, and shell scripts.

There are different types of a shell, just as there are different typess of operating systems. Each type of shell has its own set of recognized commands and functions.

A shell script is a computer program designed to be run by the Unix/Linux shell which could be one of the following:

- The Bourne Shell
- The C Shell
- The Korn Shell
- The GNU Bourne-Again Shell

Shell Prompt

The prompt, $, which is called the command prompt, is issued by the shell. While the prompt is displayed, we can type a command.

Shell Types:

In Unix, there are two major types of shells −

- Bourne shell − If we are using a Bourne-type shell, the $ character is the default prompt.
- C shell − If we are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)
- POSIX shell (sh)

The different C-type shells follow −

- C shell (csh)
- TENEX/TOPS C shell (tcsh)

The original Unix shell was written in the mid-1970s by Stephen R. Bourne while he was at the AT&T Bell Labs in New Jersey.

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Bourne shell is usually installed as /bin/sh on most versions of Unix. For this reason, it is the shell of choice for writing scripts that can be used on different versions of Unix.

The Bourne shell program name is `sh` and its path in the Unix file system hierarchy is `/bin/sh`.

The C shell, csh, was modeled on the C programming language, including the control structures and the expression grammar. It was written by Bill Joy as a graduate student at University of California, Berkeley, and was widely distributed with BSD Unix.

The C shell also introduced many features for interactive work, including the history and editing mechanisms, aliases, directory stacks, tilde notation, cdpath, job control and path hashing.

Shell Scripts

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by # sign, describing the steps.

Shell scripts and functions are both interpreted. This means they are not compiled.

Shell variables in Unix

A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.

Variable Types

When a shell is running, three main types of variables are present −

- **Local Variables** − A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.
- **Environment Variables** − An environment variable is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually, a shell script defines only those environment variables that are needed by the programs that it runs.
- **Shell Variables** − A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

An important Unix concept is the environment, which is defined by environment variables.

Unix Environment Variables

Following is the list of important environment variables. These variables are set and accessed as mentioned below –

Variable	Description
DISPLAY	Contains the identifier for the display that X11 programs should use by default.
HOME	Indicates the home directory of the current user: the default argument for the cd built-in command.
IFS	Indicates the Internal Field Separator that is used by the parser for word splitting after expansion.
PATH	Indicates the search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
PWD	Indicates the current working directory as set by the cd command.
RANDOM	Generates a random integer between 0 and 32,767 each time it is referenced.
TERM	Refers to the display type.
TZ	Refers to Time zone. It can take values like GMT, AST, etc.
UID	Expands to the numeric user ID of the current user, initialized at the shell startup.

Unix Special Variables:

The following table shows a number of special variables that can be used in your shell scripts −

Variable	Description
$0	The filename of the current script.
$n	These variables correspond to the arguments with which a script was invoked.

	Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
$#	The number of arguments supplied to a script.
$*	All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
$@	All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
$?	The exit status of the last command executed.
$$	The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
$!	The process number of the last background command.

Command-Line Arguments

The command-line arguments $1, $2, $3, ...$9 are positional parameters, with $0 pointing to the actual command, program, shell script, or function and $1, $2, $3, ...$9 as the arguments to the command.

Special Parameters $* and $@

There are special parameters that allow accessing all the command-line arguments at once. $* and $@ both will act the same unless they are enclosed in double quotes, "".

Both the parameters specify the command-line arguments. However, the "$*" special parameter takes the entire list as one argument with spaces between and the "$@" special parameter takes the entire list and separates it into separate arguments.

Basic UNIX commands

Files:

ls: lists your files

ls-l: lists your files in 'long format', which contains lots of useful information, e.g. the exact size of the file, who owns the file and who has the right to look at it, and when it was last modified.

ls-a: lists all files, including the ones whose filenames begin in a dot, which you do not always want to see.
There are many more options, for example to list files by size, by date, recursively etc.

more *filename*: shows the first part of a file, just as much as will fit on one screen. Just hit the space bar to see more or q to quit.

emacs *filename*: is an editor that lets you create and edit a file.

mv *filename1 filename2*: moves a file (i.e. gives it a different name, or moves it into a different directory.

cp *filename1 filename2*: copies a file

rm *filename*: removes a file. It is wise to use the option rm i, which will ask you for confirmation before actually deleting anything.

diff *filename1 filename2*: compares files, and shows where they differ

***wc filename*:** tells you how many lines, words, and characters there are in a file

chmod *options filename*: lets you change the read, write, and execute permissions on your files. The default is that only you can look at them and change them, but you may sometimes want to change these permissions. For example, **chmod o+r *filename*** will make the file readable for everyone, and **chmod or *filename*** will make it unreadable for others again.

File Compression:

gzip *filename*: compresses files, so that they take up much less space. Usually text files compress to about half their original size, but it depends very much on the size of the file and the nature of the contents. There are other tools for this purpose, too (e.g. compress), but gzip usually gives the highest compression rate.

unzip *filename*: uncompresses files compressed by gzip.

Printing:

lpr *filename*: print. Use the P option to specify the printer name if you want to use a printer other than your default printer.

lpq: check out the printer queue, e.g. to get the number needed for removal, or to see how many other files will be printed before yours will come out

lprm *jobnumber*: remove something from the printer queue. You can find the job number by using lpq

genscript: converts plain text files into postscript for printing, and gives you some options for formatting.

dvips *filename*: print .dvi files

Directories:

Directories are used to group files together in a hierarchical structure.

mkdir *dirname*: make a new directory

cd *dirname*: change directory. You always start out in your 'home directory', and you can get back there by typing 'cd' without arguments.

pwd: tells you where you currently are.

Finding things:

ff: find files anywhere on the system.

grep *string filename(s)*: looks for the string in the files. This can be useful a lot of purposes, e.g. finding the right file among many, figuring out which is the right version of something, and even doing serious corpus work.

About other people:

w: tells you who's logged in, and what they're doing.

who: tells you who's logged on, and where they're coming from.

talk *username*: lets you have a (typed) conversation with another user write *username* lets you exchange online messages with another user

elm: lets you send email messages to people around the world (and, of course, read them).

About your (electronic) self:

whoami: returns your username.

passwd: lets you change your password

ps u *yourusername:* lists your processes *yourusername* lists your processes.

kill *PID*: kills (ends) the processes with the ID you gave.

du *filename:* shows the disk usage of the files and directories in *filename*

last *yourusername*: lists your last logins.

Connecting to the outside world:

nn: allows you to read news.

rlogin *hostname*: lets you connect to a remote host

telnet *hostname:* also lets you connect to a remote host. Use rlogin whenever possible.

ftp *hostname*: lets you download files from a remote host which is set up as an ftpserver.

Lynx: lets you browse the web from an ordinary terminal.

Miscellaneous tools:

webster *word*: looks up the word in an electronic version of Webster's dictionary and returns the definition(s)

date: shows the current date and time.

cal: shows a calendar of the current month. Use e.g., '**cal 11 2020**' to get that for November 2020, or '**cal 2020**' to get the whole year.

man *commandname*: shows you the manual page for the command

For any command information use the man command_name (man stands for manual)
Type man commandname on the terminal and get the information of that command

Shell Programming:

The shell programming language incorporate most of the features that most modern programming languages offer. Shell program is series of Unix commands. Shell script can take input from user, file and output them on screen.

How to write shell script
Following steps are required to write shell script:
(1) Use any editor like *vi* or *gedit* to write shell script.
(2) After writing shell script set execute permission
(3) Execute shell script as
syntax:

```
bash shell-script-name
sh shell-script-name
./your-script-name
```

Examples:

```
$ bash bar
$ sh bar
$ ./bar
```

Prog1: Write following shell script, save it, execute it and note down the it's output.

```
$ vi ginfo
#
#
# Script to print user information who currently login , current date & time
#
clear
echo "Hello $USER"
echo "Today is \c ";date
echo "Number of user login : \c" ; who | wc -l
echo "Calendar"
cal
exit 0
```

Shell Keywords:
Keywords are the words whose meaning has already been explained to the shell. Following is the list of keywords available in the Bourne shell:

echo	If	until	trap	read	Else	case	wait	set	fi
esac	Eval	unset	while	break	Exec	readonly	do	ulimit	shift
done	Exit	umask	export	for	return				

Variables in Shell
To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold the data. Programmer can give a unique name to this memory location/address called memory variable or variable (Its a named storage location that may take different values, but only one at a time).

In Linux (Shell), there are two types of variable:
(1) **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
(2) **User defined variables** (UDV) - Created and maintained by user. This type of variable defined in lower letters.

How to define User defined variables (UDV)
To define UDV use following syntax
Syntax:
variable_name=value

'value' is assigned to given 'variable name' and Value must be on right side = sign.

Example:
```
$ no=10# this is ok
$ 10=no# Error, NOT Ok, Value must be on right side of = sign.
```
To define variable called 'vech' having value Bus
```
$ vech=Bus
```
To define variable called n having value 10
```
$ n=10
```

Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character. For e.g. Valid shell variable are as follows
HOME, SYSTEM_VERSION, vech, no

echo Command
Use echo command to display text or value of variable.

echo [options] [string, variables...]
Displays text or variables value on screen.
Options
-n Do not output the trailing new line.
-e Enable interpretation of the following backslash escaped characters in the strings:
\a alert (bell)
\b backspace
\c suppress trailing new line
\n new line
\r carriage return
\t horizontal tab
\\ backslash

For e.g. $ echo -e "An apple a day keeps away \a\t\tdoctor\n"

Shell Arithmetic
Use to perform arithmetic operations. To carry out arithmetic operations we have to use the command *expr* which is capable of evaluating arithmetic expression
Syntax:
expr op1 math-operator op2
Examples:
a = 10, b = 10
echo `expr $a + $b`
echo `expr $a – $b`
echo `expr $a \* $b`
echo `expr $a / $b`
echo `expr $a % $b`
on execute of this , get the following output

30, 10, 200, 2, 0

expr is capable of carrying out only integer arithmetic. To carry out arithmetic on real numbers it is necessary to use *bc* command
a = 10.5, b = 3.5
Example:
c = `echo $a + $b`| bc
d = `echo $a – $b'| bc
*e = echo $a * $b |bc*

Example to display two commands:
$ echo "Today is date"
Can't print message with today's date.
$ echo "Today is `date`"
It will print today's date as: ……….
The read Statement
Use to get input (data from user) from keyboard and store (data) to variable.
Syntax:
read variable1, variable2,...variableN
Example:
read a b c
echo &a &b &c
More command on one command line
Syntax:
command1;command2
To run two command with one command line.
Examples:
$ date;who
Will print today's date followed by users who are currently login. Note that You can't use
$ date who
for same purpose, you must put semicolon in between date and who command.

Control instruction in shell:
The control instructions determine the flow of control in a program.
There are four types of control instructions in shell:
(a) Sequence control instruction
(b) Selection or Decision control instruction
(c) Repetition or Loop control instruction
(d) Case control instruction

- The sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program
- Decision and Case control instructions allow the computer to take a decision to which instruction is to be executed next
- The Loop control instruction helps computer to execute a group of statements repeatedly

Bourne shell offers four decision making instructions. They are:
(a) The if-then-fi statement
(b) The if-then-else-fi statement
(c) The if-then-elif-else-fi statement (Nested if else)
(d) The case-esac statement

Syntax:

```
if control command
    then
fi
```

Ex1:

```
echo Enter source and target file
read source target
if cp $source &target
  then
     echo file copied succefully
 fi
```

Ex2:

```
if cp &source $target
    then
       echo File copied successfully
 else
       echo Failed to copy the file
  fi
```

The case-esac control instruction:

Syntax:

```
case value in
choice 1)
            do this
            and this
            ;;
choice 2)
            do this
            and this
            ;;
 choice 3)
            do this
            and this
            ;;
         *)
           do this
            ;;
  esac
```

Ex:

```
echo "Enter a number from 1 to 3"
read num
case $num in
 1)    echo you entered 1
       ;;
 2)    echo you entered 2
       ;;
 3)    echo you entered 3
       ;;
  *)   echo I said 1 to 3
        ;;
  esac
```

Loops: a loop involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied. There are three loop statements.

(a) Using a for statement
(b) Using a while statement
(c) Using a until statement\

Syntax of while:

```
while control command
 do
     command1
     command 2
 done
```

Ex: *count =1*

```
while [$count –le 3]
   do
        echo "Enter the values of P, n, and r\c"
        read P n r
        si = `echo $P \* $n \* $r / 100 | bc`
         echo Simple interest = Rs $si
         count = `expr $count = 1`
     done
```

The until loop:

Syntax of until loop:

```
until control command doesn't return true
   do
          this
          and this
          and this
    done
```

There are minor differences between the working of while and until loops.

- The while loop executes till the exit status of the control command is true and terminates when the exit status becomes false.
- Unlike this the until loop executes till the exit status of the control command is false and terminates when this status becomes true

Ex:
```
i = 1
while [$i –le 10]
  do
       echo $i
       i = `expr $i + 1`
  done
```
Output: 1 2 3 4 5 5 7 8 9 10

Ex:
```
i = 1
while [$i –gt 10]
  do
       echo $i
       i = `expr $i + 1`
  done
```
Output: 1 2 3 4 5 5 7 8 9 10

The for loop:

The general term of for statement

```
for control-variable in value1 value2 value3
    do
           Command1
           Command2
           Command2
    done
```

Ex:
```
for word in High on a hill was a lovely mountain
do
     echo $word
done
```

The word in the for loop is known as control variable

Output:
```
High
on
a
hill
was
a
lovely
mountain
```

Command line argument

```
for word in $*
 do
        echo $word
 done
```

The ***test*** command:

The test command can carry out several types of tests. These are:

(a) Numerical test: used when comparisons between two numbers
(b) String test: handle the string test
(c) File test: for checking the status of file

Operator of numerical test:

Operator	**Meaning**
-gt	Greater than
-lt	Less that
-ge	Greater than or equal to
-le	Less than or equal to
-ne	Not equal to
-eq	Equal to

Operator of string test

Operator	**Meaning**
string1 = string2	True if the strings are same
string1!= string2	True if the strings are different
-n string	True if the length of the string is greater than zero
-z string	True if the length of the string is zero
string	True if the string is not a null string

Operator of file test

Operator	**Meaning**
-s file	True if the file exist and has a size greater than 0
-f file	True if the file exist and is not a directory
-d file	True if the file exist and is a directory file
-c file	True if the file exist and a character special file
-b file	True if the file exist and a block special file
-r file	True if the file exist and has a read permission to it
-w file	True if the file exist and has a write permission to it
-x file	True if the file exist and has a execute permission to it

Logical Operators:

Shell allows usage of three logical operators while performing a test. These are;

(a) –a (read as AND)
(b) –o (read as OR
(c) ! (read as NOT)

Ex: The marks obtained by a student in 5 different subjects are input through the keyboard. Write a program to calculate the division obtained by the student. The student gets a division as per the following rules:

Percentage above or equal to 60 – First division
Percentage between 45 and 59 – Second division
Percentage between 33 and 44 – Third division
Percentage less than 33 – Fail

Program:

```
echo “Enter marks in five subjects\c”
read m1 m2 m3 m4 m5
per = `expr \($m1 + $m2 + $m3 + $m4 + $m5\) / 5`
if [per –ge 60]
 then
    echo First division
      elif [per –ge 45 –a per –lt 60]
          then
             echo Second division
                  elif [per –ge 33 –a per –lt 45]
                      then
                         echo Third division
                             else
                                   echo Fail
                         fi
                  fi
           fi
```

Program1: Write a shell script to find the smallest of three numbers

```
echo Enter three numbers
read a b c
s = $a
if [$b –lt $s]
  then
     s = $b
fi
if [$c –lt $s]
  then
      s = $c
fi
echo Smallest of $a $b and $c is $s
```

Program2: Write a shell program to find the largest of three numbers

```
echo Enter three numbers
read a b c
l = $a
if [$b –gt $l]
  then
     l = $b
fi
if [$c –lt $l]
  then
      l = $c
fi
echo Largest of $a $b and $c is $l
```

Program3: Write a shell script to find the factorial of given number

```
echo Enter a number
read num
n = $num
fact = 1
while [$num –ge 1]
  do
     fact = `expr $fact \* $n`
     n = `expr $n– 1
 done
echo Factorial of $num is $fact
```

Program4: Write a shell program to generate n terms of Fibonacci Series.

```
echo Enter terms
read n
x = 0 y = 1 i = 2
echo Fibonacci Series upto $n terms
echo “$x”
echo “$y”
while [$i –lt $n]
do
   then
       z = `expr $x + $y`
        i = `expr $i + 1`
        echo $z
         x = $y
         y = $z
  done
```

Shell Metacharacters:

Sometimes the metacharacters are also called regular expression. All the metacharacters can be classified as:

Type	**Metacharacters**
Filename substitution	? * [….] [!...]
I/O Redirection	< > >> << m> m>&n
Process execution	; ( ) & && ||
Quoting Metacharacters	\ “ ” ‘ ‘ ` `
Positional parameters	$1 ….. $9
Special parameters	$0 $* $@ $# $! $$ $-

Special parameters:

Metacharacters	**Meaning**
$$	PID of current shell
$?	Exit status of the last executed command
$!	PID of last background process
$-	Current shell settings
$#	Total number of positional parameters
$0	Name of the command being executed
$*	List of all shell arguments. Cant yield each argument separately
$@	Similar to $*, but yields each argument separately when enclosed in double quotes

Array Basics in Shell Scripting
An array is a systematic arrangement of the same type of data. But in Shell script Array is a variable which contains multiple values may be of same type or different type since by default in shell script everything is treated as a string. An array is zero-based ie indexing start with 0.

An array can be declare in a shell script in different ways.

1. Indirect Declaration: In this declaration, assigned a value in a particular index of Array Variable. No need to first declare. *ARRAYNAME[INDEXNR]=value*
2. Explicit Declaration: In this declaration, first declare array then assigned the values. *declare -a ARRAYNAME*
3. Compound Assignment: In this assignment, we declare array with a bunch of values. We can add other values later too. *ARRAYNAME=(value1 value2 .... valueN)*

To Print Array Value in Shell Script?

To Print All elements
[@] & [*] means All elements of Array.

echo ${ARRAYNAME[*]}

```
# To declare static Array
arr=(prakhar ankit 1 rishabh manish abhinav)

# To print all elements of array
echo ${arr[@]}
echo ${arr[*]}
echo ${arr[@]:0}
echo ${arr[*]:0}
```

Output:

```
prakhar ankit 1 rishabh manish abhinav
prakhar ankit 1 rishabh manish abhinav
prakhar ankit 1 rishabh manish abhinav
prakhar ankit 1 rishabh manish abhinav
```