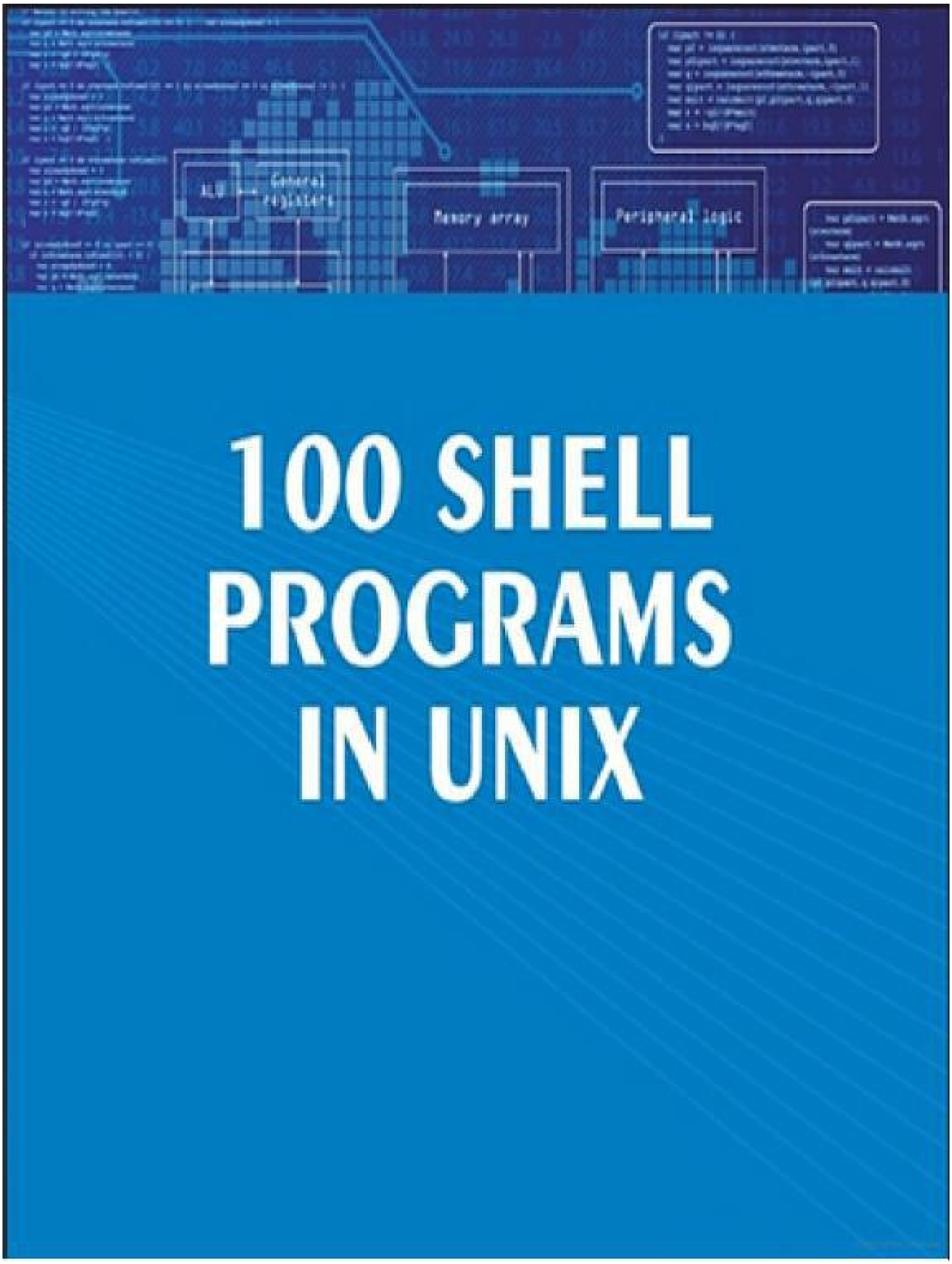
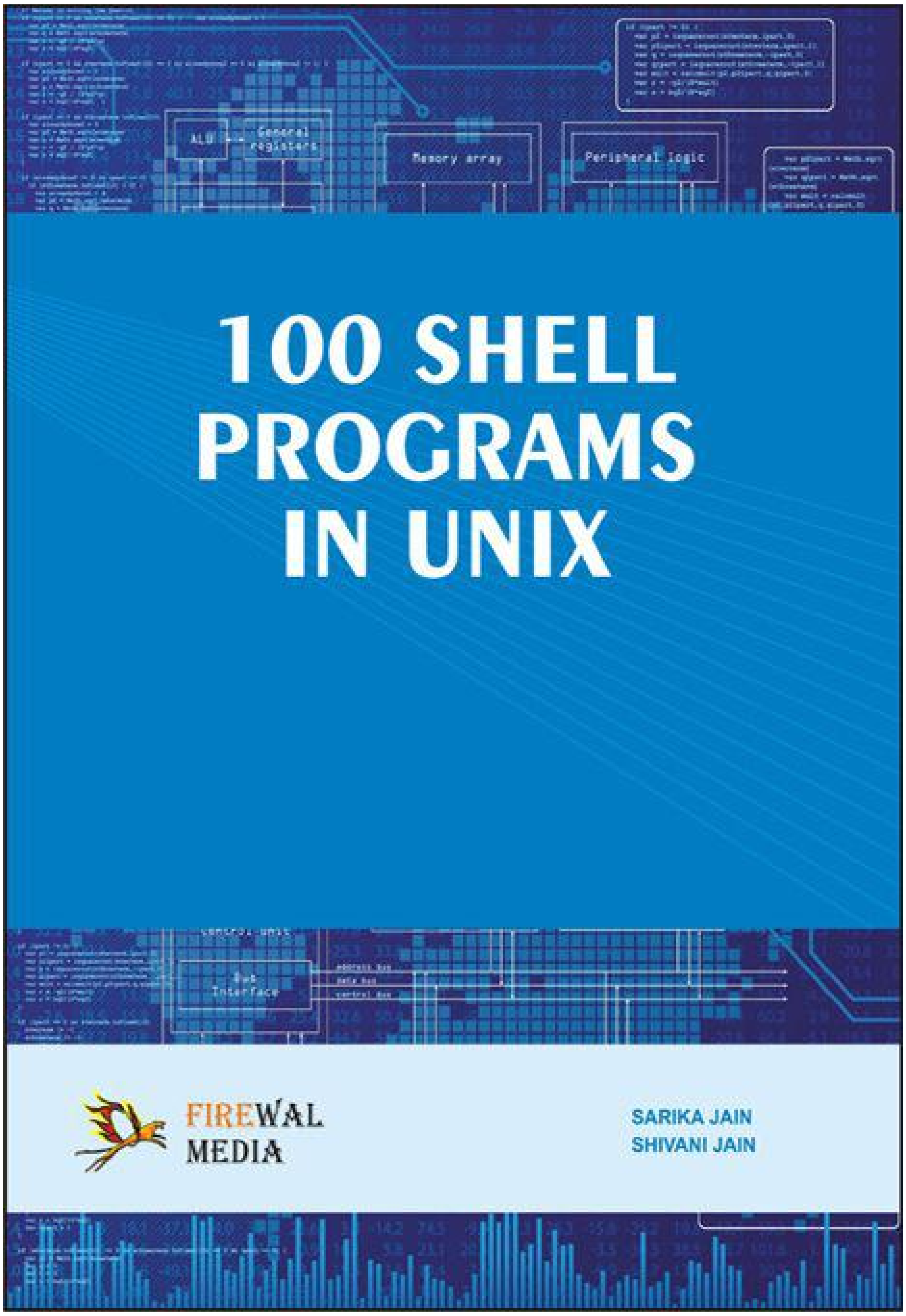
Sarika Jain
100 Shell Programs in Unix





100 SHELL PROGRAMS

IN UNIX

By
Sarika Jain

Head, MCA Department

ABSS, Meerut (U.P.)
Shivani Jain

Senior Lecturer, Department of Computer Science

VCE, Meerut (U.P.)

FIREWAL MEDIA
(An Imprint of Laxmi Publications Pvt. Ltd.)
BANGALORE . CHENNAI . COCHIN . GUWAHATI . HYDERABAD

JALANDHAR . KOLKATA . LUCKNOW . MUMBAI . RANCHI

NEW DELHI
Published by :

FIREWAL MEDIA

(*An Imprint of Laxmi Publications Pvt. Ltd.*)

113, Golden House, Daryaganj,

New Delhi-110002
Phone : 011-43 53 25 00

Fax : 011-43 53 25 28
www.laxmipublications.com

info@laxmipublications.com

First Edition
: 2009
OFFICES

☎

Bangalore
080-26 61 15 61
☎

Chennai
044-24 34 47 26
☎

Cochin
0484-239 70 04
☎

Guwahati
0361-254 36 69, 251 38 81
☎

Hyderabad
040-24 65 23 33
✆

Jalandhar
0181-222 12 72
✆

Kolkata
033-22 27 43 84
✆

Lucknow
0522-220 95 78
✆

Mumbai
022-24 91 54 15, 24 92 78 69
✆

Ranchi
0651-221 47 64
FSH-3030-100-100 SHELL PROG IN UNIX
C–17102/09/02
Typeset at
: Kalyani Computer Services, New Delhi.
Printed at
: Mehra Offset Press, Delhi.

CONTENTS

P REFACE

The UNIX system is so successful. Why? *First* , because UNIX is portable, i.e., runs on a range of computers and adapts to particular requirements. *Second* , the UNIX programming environment is unusually rich and productive. The UNIX system has become very popular, and there are number of versions in wide use. Regardless of the version you run on your system, the difference in coding you find will be minor.

The book's small size is meant to keep your investment in time down to a minimum but with the greatest possible amount of knowledge. This book is organized as follows: Part I is an introduction to the most basic use of the system. It covers logging in, the file system, commonly used commands, and logging out. Part II contains 100 programs (including shell script and programs in C).

The best way to learn something is by doing it. Kindly practice the programs and verify or contradict what we say. All the examples in this text are actual, runnable code tested on UNIX system.

As a reader of this book, you are the most important critic and commentator. You can email or write to us directly to let us know what you did or didn't like about this book – as well as what we can do to make our book stronger.

–AUTHORS

A CKNOWLEDGEMENT

We are grateful to many people for constructive comments and criticisms, and for their help in improving our code. The work of an author is only as good as the support from their family members and friends. Sarika Jain would like to specially thank her husband, Anuj Jain for letting her off all her household chores while working in the tree house on this project. Likewise, Shivani Jain wants to thank her all family members for their understanding and encouragement throughout this project. We cound not have done this without all of you.

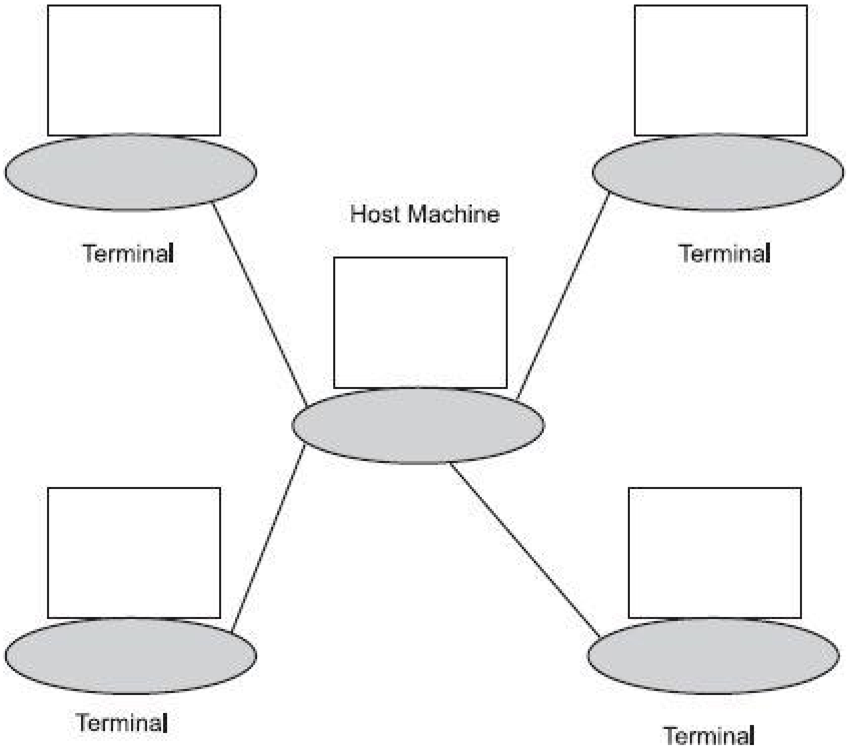
–AUTHORS

PART I

I NTRODUCTION

I. AN OVERVIEW

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs, which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops. UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows, which provides an easy to use environment. UNIX (and Linux, which is Linus Torvald's version of UNIX) has deep roots in the computer industry. UNIX is a very powerful multitasking and multi-user system. Multitasking means a user can run multiple programs simultaneously with in one single login of the system. Multi-user means that many users can simultaneously and securely use the same machine with their separate dumb terminals. The following figure shows a typical UNIX setup:



II. SALIENT FEATURES OF UNIX

Among many salient features the UNIX offers, few are listed below:

Multi-user
Multitasking
Communication
Security
Portability
Capability
Time sharing
Command interpreter & background processing
Hierarchical file system
Dos-Unix interface
Simple command
System administration & job accounting
Tools & utilities
Shell programming
Availability of 4GL and RDBMS
Library of application packages

III. HARDWARE REQUIREMENTS FOR UNIX

There are some prerequisites for a system that can host and take best advantage of UNIX. These are a PC/AT or higher with an 80 MB hard disk and at least 4MB of RAM on a 16-bit microprocessor (80286/80386/80486). The dumb terminals are connected to the host machine through a 4/8/16 port controller card installed in the expansion slot on the motherboard of the host machine. More the number of terminals more should be the memory on the host machine. Out of 80 MB disk space, almost 40MB is eaten away by the actual UNIX OS files and another 10-20 MB is used as swap space. For each terminal to be supported, 0.75 to 1 MB should be present in the host machine.

IV. GETTING STARTED

A system administrator supervises the working of UNIX on any installation. In UNIX, there are different types of accounts. The root account is the administrator user account. It has the most privileges available to the system. Then are individual user accounts having far fewer privileges. Access to many user accounts can also be controlled at once by assigning users to groups. UNIX is case sensitive and is strongly oriented towards devices with lower case. The system administrator configures every individual on the system and supplies them with credentials (username and password).

1. Logging-in

Given that your terminal is connected to the host computer and is powered on, the display prompts you for your login name.

login:

When you get the login: message, type your login name. Follow it by pressing RETURN, after which you receive the password prompt.

password:

At this stage, you must type in your password.

You get three to five attempts to get the login – password combination right before your terminal is disconnected. Once you successfully login, you get a *prompt,* usually a single character, indicating that the system is ready to accept commands from you. The prompt is most likely to be a dollar sign ($) (for Bourne Shell), or a percent sign (%) (for C Shell), but you can change it to anything you like.

2. Typing Commands

On receiving the prompt, commands can be typed. When you see the prompt ($), type who am I and press RETURN.

```
$ who am i
tom tty 3a Jul 18 10:10
```

The system should reply with your user name, system's name, and when the user logged on. If you make a mistake typing the name of a command, you will be told that no such command exists:

```
$ today's date
today's date: not found
```

You have two ways to recover from your typing mistakes, provided you see them before you press RETURN:

(*i*) You type the *line kill* character (@). It kills the whole line and you can type the whole line again.

(ii) Use erase characters one at a time using #. Each # erases the last character typed. For example,

```
$ who a i@
who am i
tom tty3a Jul 18 10:10
```

```
$ www##ho aa#mi# i
tom tty3a Jul 18 10:10
```

3. Some Special Keys

RETURN key – The RETURN key signifies the end of a line of input. On any terminal, RETURN has a key of its own, or return may be typed by holding down the control key and typing a ‘m’.

DELETE: The DELETE key stops a program/command immediately, without waiting for it to finish. DELETE can be achieved equivalently with ctrl-c.

Ctrl-s: Ctrl-s pauses the output and the program is suspended until you start it again.

Ctrl-q: Ctrl-q resumes the program paused by ctrl-s.

Ctrl-g: rings a bell on the terminal.

Ctrl-h: can be used for backspace.

Ctrl-I: can be used for tab (eight spaces on UNIX system).

4. Logging out

Logout marks the end of a UNIX session. A user can log out by either typing ctrl-d or exit at the prompt.

V. UNIX ARCHITECTURE

Figure below shows the three layers of UNIX operating system. On the outermost layer sits the user with application programs and other utilities. The kernel interacts with the actual hardware. The shell acts as the command interpreter between the user and the kernel.

The Kernel

At the center of the UNIX onion is a program called the kernel. The kernel of UNIX is the hub of the operating system. The kernel has various functions. It manages files, carries out all the data transfer between the file system and the hardware, and also manages memory. It allocates time and memory too.

The Shell

The shell acts as an interface between the user and the kernel. Shell is an intermediate program that accepts the commands, typed at the terminal and executes them to obtain kernel understandable command set. Important features are given below:

Programming Language Constructs : Shell provides powerful constructs using which exiting commands can be used to frame the job sequences or develop new utilities.

Shell Scripts : Shell commands and constructs are stored in a file, which can later be used to execute these commands like a program. This file is generally called shell script.

Shell Variables : Like other programming language one can define variable in shell program also. These variables are identified by prefixing their names with ‘$’ sign.

Variables can be classified into four categories:

Standard Variables: These are predefined in the system and therefore called built-in-variables. They hold certain system information related to particular user environment.

These are also called environmental variables.

PS1 : Represent the first prompt of the user, Default is ‘$’.

```
$echo $PS1
[\u@\h \W]\$
$PS1=#
#
```

PS2 : Represent the second prompt of the user, Default is ‘>’.

```
$echo $PS2
>
$
```

Logname: User’s login name

```
$logname
```

```
root
$
```

Positional Parameters : These are variables which receives their values from command-line as an argument. These are identified by their position on the command line as $0, $1, $2,… where $0 holds name of program and others denote another command-line argument.

Special Shell Variable : These are the variables that hold certain other information such as $$ stores process ID of the current shell, $! Stores process ID of last back ground process.

User Defined Variables: Users may define their own variables either in shell script or at the shell prompts. These variables can have any name except for those described above.

UNIX supports different types of shells. Some of these shells are:

Bourne Shell (sh)
C Shell (csh)
Korn Shell (ksh)
Job Shell (jsh)

VI. UNIX BASIC COMMANDS

A text editor is a program for storing and manipulating information in the computer. Three of the most popular editors in UNIX system are ed, vi and emacs. The ed editor works on any terminal as it takes no advantage of special terminal features.

Vi. Editor

Vi, (stands for Visual Editor,) is one of the most significant tools provided by UNIX and is used to create and edit text files. It is a screen-oriented editor that is extremely fast when scrolling through large documents. It does not support any document formatting like bold/italics, spell checking or any views of a document, as it will look when printed.

Table: Commands for quiting vi

Commands

Functions

ZZ

Write the buffer to the file and quits vi.

:wq

Write the buffer to the file and quits vi.

:w filename

Write the buffer to the file filename (new).

:q

Quits vi if changes made to the buffer were written to a file.

:w!

Overwrites the existing file filename with the contents of the buffer.

:q!

Quits vi whether or not change made to the buffer were written to a file. Does not incorporate changes made to the buffer since the last write (:w) command.

For example, addition of two numbers Steps to write a program

Step-1:

```
vi prg1
clear
echo “Input Value of a & b :”
read a
read b
c=‘expr $a + $b’
echo $c
```

Step-2:

(
i
). press Esc
(
ii
). :wq

Step-3:

sh prg1

Result:

```
Let a = 10
Let b = 12
Output is 22
```

The reader is motivated to practice the following commands with all possible options

1. The ls command

: ls [option] filename

The ls command lists the names of files in given directory or in current directory if no filename is specified. The names of files are in ascending order.

[Options]: -l : long listing about each file.

-t : files listed in order in which they were last changed, most recent first.

2.
pwd
command – pwd

3.
mkdir
– mkdir <filename>

4.
cd
– cd <filename>

5.
rmdir
– rmdir <filename>

6.
chmod
– To Assign Permissions to files.

UNIX supports two levels of security one is through login and another security is implemented by assigning different types of access permissions to different files.

UNIX divides all users into three categories:

1. Owner
2. Group
3. Others

Syntax

Chmod nnn<file>

Where n is a number from 0 to 7 representing an octal value. First n denotes the permission for owner, next n for group and the last n for others. These numbers are:

4: For Read Permission (r)
2: For Write Permission (w)
1: For Execute Permission (x)

To assign more than one permission, respective octal values are added. As to assign read and write permission, octal value will be the sum of 4 (read) and 2 (write), *i.e.,* 6. The permission set by these digits and their sum are given below:

Absolute Value
Break Up

Meaning

0

-

No permission

1

-

Only execute

2

-

Only write

3

2+1

Write & execute

4

-

Only read

5

4+1

Read & execute

6

4+2

Read & write

7

4+2+1

Read, write, execute

Examples:

$ chmod 400 <filename> : owner has only read permission.

$ chmod 700 <filename> : owner has read, write and execute permissions.

$ chmod 777 <filename> : owner, group and others have all permissions.

Another method to assigning permissions to files is symbolic method. To change permissions through this method one must specify:

Type of user (u,g,o).

Type of permission (r,w,x).

Whether the permission is to be granted(+) or revoked(-).

Name of the file.

Examples:

$ chmod u+r<file>   : Add read permissions to owner.

$ chmod a+rw<file> : Add read/write permission to all users. (a means all users)

$ chmod -w<file>   : Remove write permission from all users.

7. **mv** : Move files.

Syntax:

```
$ mv <file1> <file2>
```

8. **cp:** Copy files.

Syntax:

```
$ cp <file1> <file2>
```

9. **rm:** Remove files

Syntax:

```
$ rm <file1>
```

10. **ln:** Link files.

‘ln’ command is used for establishing an additional name for the same ordinary file so that it can be shared between two users.

Syntax:

$ ln <source> <target>

11. **find:** To Find files.

Syntax:

$ find <pathname><condition><action>

12. **cat:** To view files.

Syntax:

$ cat <filename>

13. Combine files.

Syntax:

$ cat file1.dat file2.bac file3.pqr>file4

This command merges the files (file1.dat, file2.bac and file3.pqr) into file4 to make a combined file.

14. **pr** : To Print files.

Syntax:

$pr <filename>

15. **sort:** To Sort the contents of a file.

Syntax:

$ sort <filename>

To explain this let us prepare a file:

```
$ cat temp.dat
Hyderabad
Delhi
Lucknow
Agra
Banglore
```

Now to arrange file in alphabetic order we can sort the file in this manner :

$ sort temp.dat It will display the following result on the screen

```
Agra
Banglore
Delhi
Hyderabad
Lucknow
```

16. **cmp:** To compare files.

Syntax:

$ cmp <file1> <file2> Result will look like

```
File1 file2 differ: char 280, line 18
```

Filters and Pipes

A filter is a program that takes input from the standard input, filters it and sends output to standard output. Some of filters provided by UNIX are grep, pg, wc, tr etc.

Filters and pipes commands

1.

grep

- search a file for keywords.

Syntax:

$grep regular_expr filename

Example

$grep “abc” emp.txt

2.

pg

- Used to display data one page (screenful) at a time.

Syntax:

$pg filename

3.

wc

- count number of lines/words/characters in file.

Syntax:

$wc [option] filename

Option

-l :Display number of lines

-w :Display number of words

-c :Display number of characters

4.

tr

– Translate or delete characters.

Syntax:

$tr [option] set1 set2

Example:

$tr ‘A-Z’ ‘a-z’| cat a1

5.

uniq -

The uniq command is used to display the uniq(ue) lines in a sorted file.

Syntax:

$uniq file1 file2

OTHER COMMANDS

1.

who -

list users currently logged in.

Syntax

$who

2.

tty -

Displays the terminal pathname.

syntax:

$tty

3.

echo –

display output on the screen. Echo also recognizes the c-language escape sequences that begin with a backslash.

\b : Backspace

\c : Print line without new line

\f : Form feed

\n : New-line

\r : Carriage return

\t : Tab

\v : Vertical Tab

\\ : Backslash

\nnn: It replaces the octal digits nnn to ASCII characters.

4. **ps –** Show list current processes.

Syntax:

$ps

5. **date –** display current date.

Syntax:

```
$date
```

6. **password –** Changes the password.

Syntax:

```
$passwd
```

7. **clear –** clear screen.

Syntax:

```
$clear
```

8. **cal –** display calendar.

Syntax:

```
$cal
$cal month year
$cal year
```

9. **banner –** prints the specified string in large letters.

Syntax:

```
$banner HELLO
```

10. **man -** read the online manual page for a command.

Syntax:

```
$man command
```

11. **less -** display a file a page at a time.

Syntax:

```
$less filename
```

12. **tail -** display the last few lines of a file.

Syntax:

```
$tail filename
```

13. **head -** display the first few lines of a file.

Syntax:

```
$head filename
```

14. **whatis -** brief description of a command.

Syntax:

```
$whatis command
```

15. **id –** It shows the user and group ID and corresponding name.

Syntax:

```
$id
```

16. **uname –** This command prints the name of current system on the standard output.

Syntax:

```
$uname
```

Calculator-bc (base conversion): You can call it best calculator. Once you type bc at the prompt, you are in the calculator mode. The prompt disappears.

```
$bc
10+5
15
3.8*2-4
3.6
quit
```

bc also supports functions like sqrt.

```
$bc
sqrt(16)
4
for(j=1,j<=5;j++)j
1
2
3
```

```
4
5
quit
$
```

Math related command (factor)

```
$factor 15
15: 3 5 24
24: 2 2 2 3
press ctrl+c for quiting
$
```

Taking Decisions

The if else Statement: The general format of this statement is :

```
if test <condition>
then
command(s)
else
command(s)
fi
```

The case…esac Statement: This is another decision making statement provide by UNIX. It is also known as multi conditional statement.

The general format of this statement is:

```
case $variable in
value 1) command(s);;
value 2) command(s);;
*) command ;;
esac
```

Here, the command(s) given under *) will execute when nothing matches.

The loop control structure:

The FOR loop: The FOR loop in UNIX is different from the FOR loop in other languages. Here lists of items are specified, instead of a range, say 1 to 50.

Format of this loop may be given as:

```
for variable in list
do
command
______
______
______
done
```

Example:

```
for i in 1 2 3 4 5 6 7 8 9 10 11 12
do
cal $i 2000
done
```

The WHILE loop: UNIX supports while… statements to perform repetitive tasks. The format of this statement may be given as:

```
while test <condition>
do
command(s)
done
```

or

```
while [ <condition> ]
do
```

```
command(s)
done
```

Example:

```
a=0
while test $a – le 10
do
echo $a
a=‘expr $a + 1’
done
```

The Until loop : This is another type of looping statement supported by UNIX. It has following format:

```
until [ <condition> ]
do
command(s)
done
```

The functioning of UNTIL loop is inverse of WHILE loop. Here, the set of specified command(s) are executed as long as the condition is false.

Break and continue statements: Both of these commands are used with WHILE, UNTIL and FOR loops. The break statement terminates the inner-most loop and executes the statements, written after it.

On the other hand, ‘continue’ does not terminate the loop, but continues the execution of the next cycle of the loop. Both of these can be illustrated as:

The break Statement:

```
while true
do
——
——
if——
then
break
else
——
——
fi
——
——
done
command(s)
```

The continue Statement:

```
while true
do
——
——
if——
then
continue
fi
done
command(s)
```

VII. PORTABILITY WITH C

To write a ‘C’ program in UNIX, type your program in vi editor. Give the file name with extension .c (prg1.c). cc is the compiler of ‘C’ in UNIX.

Example:

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
int a=10,b=20,c;
c=a+b;
printf(“sum of a & b=%d”,c);
return 0;
}
```

Procedure:

Let’s start by compiling the program prg1.c. Type:

```
cc prg1.c
```

If you list files in the current directory, you will find that this command has produced a file called a.out. You can then run it by typing:

```
./a.out
```

Now let’s suppose that you want to give your executable program the name “sumofab”, rather than the default (and not very informative) name “a.out”. Of course, you could just rename the file, but it is better to use the “-o” option to tell cc what filename to create.

```
cc -o sumofab prg1.c
```

And now, to run it you would type:

```
./sumofa
```

PART II

I PROGRAMS

1. Write a shell script to find whether an input integer is even or odd.

```
$vi prg1
clear
echo “enter a number”
read x
y=‘expr $x % 2’
if test $y –eq 0
then
echo “Number is even” else
echo “Number is odd” fi
```

Sample Run

$sh prg1

```
enter a number
11
```

```
Number is odd
$sh prg1
enter a number
12
Number is even
```

2. Write a shell script to find out the greatest among three inputs.

```
$vi prg2
clear
echo “enter the value of a b c”
read a
read b
read c
if test $a –gt $b –a $a –gt $c
then
echo “a is greatest”
else
if test $b –gt $c
then
echo “b is greatest”
else
echo “c is greatest”
fi
fi
```

Sample Run

```
$sh prg2
enter the value of a b c
23
33
44
c is greatest
$sh prg2
enter the value of a b c
23
55
44
b is greatest
$sh prg2
enter the value of a b c
78
33
44
a is greatest
```

3. Write a shell script to calculate the net salary of an employee in a particular month considering various allowances (TA, DA, HRA) and deductions (INCOME TAX, PROVIDEND FUND) as:

(

a

)

TA=15 percent of basic salary

(

b

)

DA=2 percent of basic salary

(

c

)

HRA=10 percent of basic salary

(

d

)

INCOME TAX=5 percent of salary

(

e

)

PROVIDEND FUND=10 percent of salary

$vi prg3

```
clear
echo “enter basic salary”
read bs
hra=‘echo $bs \* 10 / 100 | bc’
ta=‘echo $bs \* 15 / 100 | bc’
da=‘echo $bs \* 2 / 100 | bc’
tax=‘echo $bs \* 5 / 100 | bc’
pf=‘echo $bs \* 10 / 100 | bc’
add=‘echo $hra + $ta + $da | bc’
ded=‘echo $tax + $pf | bc ‘
netsal=‘echo $bs + $add - $ded | bc’
echo
echo net salary is $netsal
```

Sample Run

$sh prg3

```
enter basic salary
2240
net salary is 2540
```

4. A departmental store announces its festival scheme to customers on cash payment. The scheme is as follows-

(

a

) If purchase amount is less than 1000 then Tax=2% and discount=10%.

(

b

) If purchase amount is greater than 1000 then Tax=5 % and discount=20%.

$vi prg4

```
clear
echo “enter purchase amount”
```

```
read pa
if [ $pa –lt 1000 ]
then
tax=‘echo $pa \* 2 /100 | bc’
discount=‘echo $pa \* 10 / 100 | bc’
else
tax=‘echo $pa \* 5 /100 | bc’
discount=‘echo $pa \* 20 / 100 | bc’ fi
amount=‘expr $pa + $tax - $discount’ echo cash payment =$amount
```

Sample Run

```
$sh prg4
enter purchase amount
3000
cash payment =2550
```

5. Write a shell script to perform an arithmetic operation upon two inputs. The operation should also be input by the user.

```
$vi prg5
clear
echo “enter a and b”
read a
read b
echo “enter operation to be performed”
read op
case $op in
+) c=‘expr $a + $b’ ;;
-) c=‘expr $a - $b’ ;;
/) c=‘expr $a / $b’ ;;
\*) c=‘expr $a \* $b’ ;;
*) echo “no valid operation specified” ;;
esac
echo Result after performing operation on a and b is echo $c
```

Sample Run

```
$sh prg5
enter a and b
4
3
enter operation to be performed
+
Result after performing operation on a and b is
7
$sh prg5
enter a and b
6
5
enter operation to be performed
-
Result after performing operation on a and b is
1
$sh prg5
```

```
enter a and b
2
3
enter operation to be performed
*
Result after performing operation on a and b is
6
$sh prg5
enter a and b
4
2
enter operation to be performed
/
Result after performing operation on a and b is
2
$sh prg5
enter a and b
4
5
enter operation to be performed
f
no valid operation specified
```

6. Write a shell script to find out the length of an input string.

```
$vi prg6
clear
echo “enter string”
read str
len=‘echo $str | wc –c’
len=‘expr $len – 1’
echo “length of string = $len”
```

Sample Run

```
$sh prg6
enter string
unix
length of string = 4
```

7. Write a shell script to find whether an input year is leap year or not.

```
$vi prg7
clear
echo “enter year”
read y
k=‘expr $y % 4’
if test $k –eq 0
then
echo “leap year”
else
echo “not a leap year”
```

```
fi
```

Sample Run

```
$sh prg7
enter year
2008
leap year
$sh prg7
enter year
2009
not a leap year
```

8. Make a duplicate copy of a specified file through command-line.

```
$vi prg8
clear
echo file to be copied : $1
echo new file name : $2
if test $# -lt 2 –o $# -gt 2
then
echo invalid
exit
fi
cp $1 $2
echo copy successful
```

Sample Run

```
$sh prg8 a1.txt a1.out
file to be copied : a1.txt
new file name : a1.out
copy successful
```

9. Write a shell script to concatenate two strings input by the user.

```
$vi prg9
clear
echo “enter two string
read str1
read str2
str3=‘echo $str1 $str2’
echo After concatenate : $str3
```

Sample Run

$sh prg9

```
enter two string
Shell
Programming
After concatenate : Shell Programming
```

10. Write a shell script to concatenate files.

```
$vi prg10
```

```
clear
cat>f1
cat>f2
cat f1 f2 >f3
cat f3
```

11. Program for command-line parameter & special variable.

```
$ vi prg11
clear
echo the name of the program is $0
echo the first parameter : $1
echo the second parameter : $2
echo the number of parameters are : $#
echo the parameters are : $*
```

Sample Run

```
$sh prg11 a s d f g
the name of the program is prg11
the first parameter : a
the second parameter : s
the number of parameters are : 5
the parameters are : a s d f g
```

12. Generate a table of an input integer.

```
$vi prg12
clear
echo “input number :”
read x
echo
for i in 1 2 3 4 5 6 7 8 9 10
do
t=‘expr $x \* $i’
echo $t
i=‘expr $i + 1’
done
```

Sample Run

```
$sh prg12
input number
4
4
8
12
16
20
24
28
32
36
40
```

13. Write a shell script to print all the multiplication tables (up to 10) between two given numbers.

$vi prg13

```
clear
i=1
j=10
echo enter lower limit
read low
echo enter higher limit
read high
while test $low –le $high
do
echo
echo Table of $low is
echo
while test $i –le $j
do
k=‘expr $low \* $i’
echo $low \* $i   = $k
i=‘expr $i + 1’
done
i=1
low=‘expr $low + 1’ done
```

Sample Run

$sh prg13

```
enter lower limit
2
enter higher limit
4
Table of 2 is
2 * 1   = 2
2 * 2   = 4
2 * 3   = 6
2 * 4   = 8
2 * 5   = 10
2 * 6   = 12
2 * 7   = 14
2 * 8   = 16
2 * 9   = 18
2 * 10  = 20
Table of 3 is
3 * 1   = 3
3 * 2   = 6
3 * 3   = 9
3 * 4   = 12
3 * 5   = 15
3 * 6   = 18
3 * 7   = 21
3 * 8   = 24
```

```
3 * 9   = 27
3 * 10  = 30
Table of 4 is
4 * 1   = 4
4 * 2   = 8
4 * 3   = 12
4 * 4   = 16
4 * 5   = 20
4 * 6   = 24
4 * 7   = 28
4 * 8   = 32
4 * 9   = 36
4 * 10  = 40
```

14. Write a shell script to find out the n^y, where n and y must be input by the user.

```
$vi prg14
clear
echo “enter a number”
read n
echo “enter the power
read y
i=1
j=$n
while test $i –lt $y
do
j=‘expr $j \* $n’
i=‘expr $i + 1’
done
echo $j
```

Sample Run

```
$sh prg14
enter a number
4
enter the power
2
16
```

15. Write a shell script to find out the factorial of an input.

```
$vi prg15
clear
i=1
j=1
echo “enter the number”
read num
while test $i –le $num
do
k=‘expr $i \* $j’
i=‘expr $i + 1’
```

```
j=$k
done
echo Factorial of $num is $j
```

Sample Run

```
$sh prg15
enter the number
4
Factorial of 4 is 24
```

16. Write a shell script to generate the series of even number from 0 to n. 0 2 4   n

```
$vi prg16
clear
echo “enter value of n”
read n
i=0
while test $i –le $n
do
printf “ $i”
i=‘expr $i + 2’
done
echo
```

Sample Run

```
$sh prg16
enter value of n
5
0 2 4
```

17. Write a shell script to check whether an input is a prime or not.

```
$vi prg17
clear
echo “enter number”
read num
i=2
while test $i –lt $num
do
k=‘expr $num / $i’
if test $k –eq 0
then
echo “number is not prime”
exit
fi
i=‘expr $i + 1’
done
echo “number is prime”
```

Sample Run

```
$sh prg17
enter number
4
```

```
number is not prime
$sh prg17
enter number
7
number is prime
```

18. Write a shell script to generate the primes between two given numbers.

```
$vi prg18
clear
echo “enter two numbers”
read a
echo
if [ $a -eq 0 -a $a -eq 1 ]
then
a=2
fi
read b echo
while test $a -le $b do
i=2
while test $i -lt $a
do
k=‘expr $a % $i’
if test $k -eq 0
then
break
fi
i=‘expr $i + 1’
done
if [ $i -eq $a ]
then
echo $a fi
a=‘expr $a + 1’
done
```

Sample Run

```
$sh prg18
enter two numbers
22
2
3
5
7
11
13
17
19
```

19. Write a shell script to find out the sum of series 1+2+3+………….n, where n is input by the user.

```
$vi prg19
clear
echo “enter value of n”
read n
i=1
sum=0
while test $i –le $n
do
sum=‘expr $sum + $i’
i=‘expr $i + 1’
done
echo Sum of series is $sum
```

Sample Run

```
$sh prg19
enter value of n
12
Sum of series is 78
```

20. Write a shell script to generate the series 2,4,6,8,…………n, where n must be input by the user.

$vi prg20

```
clear
echo enter value of n
read n
echo
i=2
while test $i –lt $n
do
printf “ $i, “
i=‘expr $i + 2’
done
printf “ $i”
echo
```

Sample Run

```
$sh prg20
enter value of n
21
2, 4, 6, 8, 10, 12, 14, 16, 18, 20
```

21. Write a shell script to generate the series 1, 5, 2, 10, 3, 15,………….50.

```
$vi prg21
clear
a=1
b=5
while [ $b –le 50 ]
do
printf “ $a”
printf “, $b”
```

```
a=‘expr $a + 1’
b=‘expr $b + 5’
done
echo
```

Sample Run

$sh prg21

```
1, 5, 2, 10, 3, 15, 4, 20, 5, 25, 6, 30, 7, 35, 8, 40, 9, 45, 10, 50
```

22. Write a shell script to generate the series 1+1/2+1/3+……………+1/n.

```
$vi prg22
clear
echo enter value of n
read n
echo
i=2
printf “1+”
while test $i -lt $n
do
printf “1/$i+”
i=‘expr $i + 1’
done
printf “1/$i”
echo
```

Sample Run

$sh prg22

```
enter value of n
12
1+1/2+1/3+1/4+1/5+1/6+1/7+1/8+1/9+1/10+1/11+1/12
```

23. Write a shell script to generate the series ½+2/3+3/4+……………n-1/n.

```
$vi prg23
clear
echo enter value of n
read n
echo
b=1
c=2
a=1
n=‘expr $n - 1’
while test $a -lt $n
do
printf $b/$c+
b=‘expr $b + 1’
c=‘expr $c + 1’
a=‘expr $a + 1’
done
printf $b/$c
echo
```

Sample Run

```
$sh prg23
enter value of n
12
1/2+2/3+3/4+4/5+5/6+6/7+7/8+8/9+9/10+10/11+11/12
```

24. Write a shell script to find out the sum of series $1^2+2^2+3^2+$……………n^2.

```
$vi prg24
clear
echo “enter value of n”
read n
i=1
sum=0
while test $i –le $n
do
k=‘expr $i \* $i’
sum=‘expr $sum + $k’
i=‘expr $i + 1’
done
echo Sum of series is $sum
```

Sample Run

```
$sh prg24
enter value of n
10
Sum of series is 385
```

25. The XYZ construction company plans to give a 5% year-end bonus to each of its employees earning Rs. 5,000 or more per year and a fixed bonus of Rs 250 to all other employees. Print the bonus of any employee.

```
$vi prg25
clear
echo Enter Salary of  an Employee
read sal
if [ $sal -ge 5000 ]
then
bonus=‘echo $sal \* .05 | bc’
else
bonus=250
fi
echo bonus is: $bonus
```

Sample Run

```
$sh prg25
Enter Salary of an Employee
6500
bonus is: 325.00
$sh prg25
```

```
Enter Salary of an Employee
7000
bonus is: 350.00
```

$sh prg25

```
Enter Salary of an Employee
3500
bonus is: 250
```

26. Write a shell script to find out greatest among n input integers where n is to be input by the user.

$vi prg26

```
clear
echo “Enter number of integers”
read n
echo “enter value of  integer number 1”
read j
i=2
while test $i -le $n
do
echo enter value of integer number $i
read k
if [ $j -lt $k ]
then
j=$k
fi
i=‘expr $i + 1’
done
echo Greatest input is $j
```

Sample Run

$sh prg26

```
Enter number of integers
5
enter value of integer number 1
8
enter value of integer number 2
3
enter value of integer number 3
22
enter value of integer number 4
44
enter value of integer number 5
11
Greatest input is 44
```

27. Write a shell script to read an integer and print its digits in reverse order.

$vi prg27

```
clear
echo “enter any integer”
```

```
read num
b=0
while test $num -gt 0
do
a=‘expr $num % 10’
b=‘expr \( $b + $a \) \* 10’
num=‘expr $num / 10’
done
b=‘expr $b / 10’ echo reverse=$b
```

Sample Run

```
$sh prg27
enter any integer
123
reverse=321
```

28. Sort the given numbers in the given order, i.e., either in ascending or descending order.

$vi prg28

```
clear
ans=y
while test $ans = y
do
echo Enter no. of elements to be sorted
read no
echo Enter $no elements
i=1
rm sort1
while test $i –le $no
do
read n
‘echo $n >> sort1’
i=‘expr $i + 1’
done
clear
echo input order of sorting
echo 1.Ascending
echo 2.Descending
echo enter choice
read ch
clear
case $ch in
1)    sort –n sort1>file1
echo Inputted elements in Ascending order:
cat file1 ;;
1)    sort –r sort1>file1
echo Inputted elements in Descending order:
cat file1 ;;
1)    echo “Invalid Input” ;;
esac
echo continue…….y/n
```

```
read ans
done
```

Sample Run

```
$sh prg28
Enter no. of elements to be sorted
4
Enter 4 elements
3
5
2
1
input order of sorting
1.Ascending Press 1
2.Descending Press 2
enter choice
1
Inputted elements in Ascending order:
1
2
3
5
continue…….y/n
y
Enter no. of elements to be sorted
5
Enter 5 elements
4
6
1
3
3
input order of sorting
1.Ascending Press 1
2.Descending Press 2
enter choice
2
Inputted elements in Descending order:
6
4
3
3
1
continue…….y/n
n
```

29. Write a shell script to compare two strings input by the user for equality.

```
$vi prg29
clear
echo enter string1
read str1
```

```
echo enter string2
read str2
if test $str1 = $str2
then
echo strings are equal
else
echo strings are not equal
fi
```

Sample Run

```
$sh prg29
enter string1
abc
enter string2
abc
strings are equal
$sh prg29
enter string1
xyz
enter string2
abc
strings are not equal
```

30. Write a shell script to print the characters of an input string into reverse order.

```
$vi prg30
clear
echo enter any string
read str
len=‘echo $str | wc -c’
len=‘expr $len - 1’
while test $len -ne 0
do
i=‘echo $str | cut -c $len’
a=$a$i
len=‘expr $len - 1’
done
echo reverse is $a
```

Sample Run

```
$sh prg30
enter any string
programming
reverse is gnimmargorp
```

31. Write a shell script to tell whether input string is palindrome or not.

```
$vi prg31
clear
echo enter any string
read str
len=‘echo $str | wc –c’
```

```
len=‘expr $len -1’
while test $len –ne 0
do
i=‘echo $str | cut –c $len’
a=$a$i
len=‘expr $len -1’
done
if test $str = $a
then
echo String is Palindrome
else
echo String is not Palindrome
fi
```

Sample Run

```
$sh prg31
enter any string
cmc
String is Palindrome
$sh prg31
enter any string
abc
String is not Palindrome
```

32. Write a shell script to find out the location of an input character into an input string.

```
$vi prg32
clear
echo enter any string
read str
echo enter character
read c
len=‘echo $str | wc –c’
len=‘expr $len – 1’
i=1
while test $i –le $len
do
a=‘echo $str | cut –c $i’
if test $a = $c
then
echo Position=$i
fi
i=‘expr $i + 1’
done
```

Sample Run

```
$sh prg32
enter any string
Programming
enter character
g
Position=4
Position=11
```

33. Write a shell script to count the number of characters, words, spaces in a given text.

```
$vi prg33
clear
echo “enter text”
read t
w=‘expr $t | wc –w’
c=‘expr $t | wc –c’
c=‘expr $c - 1’
s=‘expr $w – 1’
echo characters = $c
echo words = $w
echo spaces = $s
```

Sample Run

```
$sh prg33
enter text
that is a table
characters = 15
words = 4
spaces = 3
```

34. Write a shell script to print Fibonacci series.

```
$vi prg34
clear
echo enter the last element of series
read n
echo
a=0
b=1
echo $a
echo $b
i=1
while test $i –lt $n
do
c=‘expr $a + $b’
if test $c –gt $n
then
exit
fi
echo $c
a=$b
b=$c
i=‘expr $i + 1’
done
```

Sample Run

```
$sh prg34
enter value of series
5
```

```
0
1
1
2
3
5
```

35. Write a shell script to translate the contents of a file into UPPER CASE, where file name is entered through command line.

```
$vi prg35
clear
if test $# -eq 0
then
echo “No argument”
exit
fi
while test $# -gt 0
do
if test –s $1
then
if test –f $1
then
cat $1 | tr a-z A-Z >$1.up
cat $1.up
fi
else
echo $1 is not a file
fi
shift
done
echo Translation successful
```

Sample Run

```
$sh prg35 file.txt
WELCOME
HELLO
Translation successful
```

In file.txt, welcome and hello are written in small letters. After running this program, welcome and hello are converted in capital letters and saved in 1.up file

36. Write a shell script to perform following tasks-

(
a
)
Display the present working directory.

(
b
)
Clear the screen.

(
c
)
Display the current date.

(
d
)
Make a directory with its sub-directory d1.

(
e
)
Change the directory to the directory having sub directory d1.

(
f
)
Create two files (say file1 & file2) within this.

(
g
)
Provide appropriate security options to these files.

(
h
)
List the contents of directory.

$vi prg36

```
(a) Pwd
(b) clear
(c) date
(d) mkdir d
cd d
mkdir d1
(e) cd d1
(f) touch file1 file2
(g) chmod 644 file1 file2
(h) Is
```

37. The marks obtained by a student in five different subjects are input through the keyboard. The student gets a division as per the following rules. (Using else's clause).

if percentage greater than or equal to 60 get First division
if percentage greater than or equal to 50 or less than 60 get Second division
if percentage greater than or equal to 40 or less than 50 get Third division
if percentage less than 40 Fail

$vi prg37

```
clear
echo enter marks of five subjects (out of 100 each)
read m1
read m2
read m3
read m4
read m5
```

```
per=‘echo \( $m1 + $m2 + $m3 + $m4 + $m5 \) /5 | bc’
echo
echo Percentage is $per
if [ $per –ge 60 ]
then
echo First division
else
if [ $per –ge 50 –a -$per –lt 60 ]
then
echo Second division
else
if [ $per –ge 40 –a $per –lt 50 ]
then
echo Third division
else
echo Fail
fi
fi
fi
```

Sample Run

```
$sh prg37
enter marks of five subjects
44
67
80
90
67
Percentage is 69
First division
$sh prg37
enter marks of five subjects
56
54
53
51
60
Percentage is 54
Second division
$sh prg37
enter marks of five subjects
46
54
41
42
46
Percentage is 45
Third division
$sh prg37
enter marks of five subjects
34
42
31
```

```
32
23
Percentage is 32
Fail
```

38. The marks obtained by a student in two different subjects are input through the keyboard. The student gets a division as per the following rules. (Using elif clause).
if percentage greater than or equal to 60 get First division
if percentage greater than or equal to 50 or less than 60 get Second division
if percentage greater than or equal to 40 or less than 50 get Third division
if percentage less than 40 Fail

$vi prg38

```
clear
echo enter marks of five subjects
read m1
read m2
read m3
read m4
read m5
per=‘echo \( $m1 + $m2 + $m3 + $m4 + $m5 \) /5 | bc’
echo
echo Percentage is $per
if [ $per –ge 60 ]
then
echo First division
elif [ $per –ge 50 –a -$per –lt 60 ]
then
echo Second division
elif [ $per –ge 40 –a $per –lt 50 ]
then
echo Third division
else
echo Fail
fi
```

Sample Run

$sh prg38

```
enter marks of five subjects
44
67
80
90
67
Percentage is 69
First division
```

$sh prg38

```
enter marks of five subjects
56
54
53
51
```

```
60
Percentage is 54
Second division
$sh prg38
enter marks of five subjects
46
54
41
42
46
Percentage is 45
Third division
$sh prg38
enter marks of five subjects
34
42
31
32
23
Percentage is 32
Fail
```

39. Write a shell script to generate first ‘n’ terms of the following sequence without using multiplication-1 2 4 8 16 32…………n.

```
$vi prg39
clear
echo enter the value of n
read n
echo
i=1
while test $i –le $n
do
echo $i
i=‘expr $i + $i’
done
```

Sample Run

```
$sh p39
enter the value of n
20
1
2
4
8
16
```

40. Write a shell script to find greatest common divisor (GCD) for two given numbers.

```
$vi prg40
clear
```

```
echo enter numbers a and b
read a b
while [ 1 ] # infinite loop
do
c=‘expr $a % $b’
if [ $c -eq 0 ]
then
echo GCD = $b
exit
fi
a=$b
b=$c
done
```

Sample Run

```
$sh prg40
enter numbers a and b 47 3 GCD = 1
```

41. Write a shell script that takes as command-line input, a number n and a word. It then prints the word n times, one word per line.

```
$vi prg41
clear
i=1
while [ $i –le  $1 ]
do
echo $2
i=‘expr $i + 1’
done
```

Sample Run

```
$sh prg41 5 Hello
Hello
Hello
Hello
Hello
Hello
```

42. Write a shell script to remove all words that occur more than once in a list.

```
$vi prg42
clear
echo enter list
cat >file1
echo uniques are :
sort –u file1>file1.out
cat file1.out
```

Sample Run

```
$sh prg42
enter list
a
c
```

```
a
b
c
Uniques are :
a
b
c
```

43. Write a shell script to take backup of all c files.

```
$vi prg43
clear
ls abc >a1
if test – a1
then
mkdir abc
cp *.c /abc
echo backup is done
fi
```

44. Write a program in UNIX to accept range of months and display calendar within that range.

$vi prg44

```
clear
echo enter lower limit
read llimit
echo enter upper limit
read ulimit
echo enter year
read y
echo
while test $llimit -le $ulimit
do
cal $llimit $y
llimit=‘expr $llimit + 1’
done
```

Sample Run

$sh prg44

```
enter lower limit
2
enter upper limit
3
enter year
2008
February 2008
Su Mo Tu We Th Fr Sa
1  2
3  4  5  6  7  8  9
10 11 12 13 14 15 16
```

```
17 18 19 20 21 22 23
24 25 26 27 28 29


March 2008
Su Mo Tu We Th Fr Sa
1
2  3  4  5  6  7  8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

45. Write a program in UNIX to accept a year and months in that year and display calendar of those months.

$vi prg45

```
clear
echo enter month value in numeric
read m
echo enter year
read y
echo
for i in $m
do
cal $i $y
done
```

Sample Run

$sh prg45

```
enter month value in numeric
1 3 12
enter year
2008
January 2008
Su Mo Tu We Th Fr Sa
1  2  3  4  5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31


March 2008
Su Mo Tu We Th Fr
Sa
1
2  3  4  5  6  7  8
9 10 11 12 13 14
15
16 17 18 19 20 21
22
```

```
23 24 25 26 27 28
29
30 31
December 2008
Su Mo Tu We Th Fr Sa
1  2  3 4 5  6
7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

46. To find out the sum of squares of integers from m to n where m, n are input by user.

```
$vi prg46
clear
echo enter value of m and n
read m
read n
echo
s=0
while [ $m -le $n ]
do
a=‘expr $m \* $m’
s=‘expr $s + $a’
m=‘expr $m + 1’
done
echo $s
```

Sample Run

$sh prg46

```
enter value of m and n
2
5
54
```

47. To find out the greatest and smallest element of an array.

```
$ vi prg47
clear
echo Enter size of array
read no
i=0
echo
echo Enter $no elements
while [ $i -lt $no ]
do
read n[$i]
i=‘expr $i + 1’
done
high=${n[0]} low=${n[0]} k=1
while [ $k -lt $no ]
```

```
do
if [ $high -lt ${n[$k]} ]
then
high=${n[$k]}
fi
if [ $low -gt ${n[$k]} ]
then
low=${n[$k]}
fi
k=‘expr $k + 1’
done
echo highest=$high
echo lowest=$low
```

Sample Run

$sh prg47

```
Enter size of array
5
Enter 5 elements
3
22
1
55
4
highest=55
lowest=1
```

48. Write a shell script to find out whether a file is writable or not. File name must be input by the user through command-line.

$vi prg48

```
clear
if test -w $1
then
echo file is writable
else
echo file is not writable
fi
```

Sample Run

```
$sh prg48 a1.txt
file is writable
```

49. Write a program for Bubble sorting.

```
$vi prg49
clear
echo enter any no
read no
i=0
k=0
while [ $i -lt $no ]
```

```
do
read n[$i]
i=‘expr $i + 1’
done
while [ $k -lt $no ]
do
j=0
while test $j -lt $no
do
if test ${n[$k]} -lt ${n[$j]}
then
m=${n[$k]}
n[$k]=${n[$j]}
n[$j]=$m
fi
j=‘expr $j + 1’
done
k=‘expr $k + 1’
done
a=0
echo Array after bubble
sort while test $a -lt $no
do
echo “${n[$a]}” a=‘expr
$a + 1’
done
```

Sample Run

$sh prg49

```
enter any no
5
6
4
1
9
7
Array after bubble sort
1
4
6
7
9
```

50. Write a shell script to find out what type of character you have entered such as capital letter, small letter, digit, special symbol and whether you entered more than one character.

$vi prg50

```
clear
echo enter character
read char
case $char in
```

```
[A-Z]) echo you entered a capital letter ;;
[a-z]) echo you entered a small letter ;;
[0-9]) echo you entered a digit ;;
?) echo you entered a special symbol ;;
*) echo you entered more than one character ;; esac
```

Sample Run

```
$sh prg50
enter character
a
you entered a small letter
enter character
1
you entered a digit
enter character
#
you entered a special symbol
enter character
asd123
you entered more than one character
enter character
A
you entered a capital letter
```

51. Write a script that has this output:

```
Give me a U!
U
Give me a N!
N
Give me a I!
I
Give me a X!
X
```

```
$vi prg51
clear
for i in U N I X
do
echo Give me a $i!
echo $i done
```

Sample Run

```
$sh prg51
Give me a U!
U
Give me a N!
N
Give me a I!
I
Give me a X!
X
```

52. Rewrite the Q. 51 so that it uses command-line input to provide the spell out letters.

```
$sh prg52
clear for i
do
echo Give me a $i!
echo $i
done
```

Sample Run

```
sh prg52 B O O K
Give me a B!
B
Give me a O!
O
Give me a O!
O
Give me a K!
K
```

53. Write a shell script that presents a multiple-choice question, gets the user's answer, and reports back whether the answer is right, wrong, or not one of the choices.

```
$vi prg53
clear
echo UNIX is
echo a\) a Turkish Assistant Manager\'s club
echo b\) a United Nations organization
echo c\) a computer operating system
echo d\) all of the above
read answer
case $answer in
a) echo Wrong – the answer is c ;;
b) echo Wrong – the answer is c ;;
d) echo Wrong – the answer is c ;;
c) echo Right ;;
*) echo Not one of the choices ;;
esac
```

Sample Run

```
$sh prg53
UNIX is
a) a Turkish Assistant Manager's club
b) a United Nations organization
c) a computer operating system
d) all of the above a Wrong – the answer is c)
$sh prg53
UNIX is
a) a Turkish Assistant Manager's club
b) a United Nations organization
c) a computer operating system
d) all of the above c
Right
```

54. Write a shell script which accepts the word oak as an answer regardless of whether upper-case or lower-case letters are used anywhere in the word.

```
$sh prg54
clear
echo What kind of tree bears acorns\?
read response
case $response in
Oo) echo $response is correct ;;
Aa) echo $response is correct ;;
Kk) echo $response is correct ;;
*) echo sorry, that is wrong
esac
```

Sample Run

$sh prg54

```
What kind of tree bears acorns?
Aa
Aa is correct
$sh prg54
What kind of tree bears acorns?
AA
sorry, that is wrong
```

55. Write a shell script that takes a login name (say X) as a command-line argument and reports to you when that person has logged in or not. If the user wants to send a greeting to that person (X) redirection can be used to his or her terminal. (Such a script would be run in background.)

In case admin is not login, it repeatedly says "admin is not logged in" press ctrl+c

$vi prg55

```
clear
until who | grep $1 > /dev/null
do
echo sleep now
echo admin is not logged in
echo press ctrl+c
sleep 300
done
set ‘who | grep $1’ echo $1 has logged in on $2 echo hi, $1 > /dev/$2
```

Sample Run

$sh prg55 tomcat

here tomcat is the user name who is to be searched for being log in. If tomcat is not log in then the set command returns error.

```
tomcat has logged in on tty1
```

[tomcat@localhost ~]$ hi, tomcat

This output is displayed on tomcat's terminal

$sh prg55 admin

```
sleep now
admin is not logged in
press ctrl+c
```

56. Write a shell script that takes a command-line argument and reports whether it is a directory, a file, or something else.

```
$vi prg56
clear for name
do
if test -d $name
then
echo $name is a directory
elif test -f $name
then
echo $name is a file
else
echo I don\'t know what $name is
fi
done
```

Sample Run

```
$sh prg56 mnt
mnt is a directory
$sh prg56 emp.dat
emp.dat is a file
```

57. Write a shell script that asks for the capital of India and repeats the question until the user gets it right. Enter capital in small letters.

```
$vi prg57
clear
echo What is the capital of India
read ans
while test $ans != delhi
do
echo No, that\'s not it. Try again.
read ans
done
echo That is correct.
```

Sample Run

```
$sh prg57
What is the capital of India
delhi
That is correct.
$sh prg57
What is the capital of India
mumbai
No, that's not it. Try again.
```

58. Write a number-guessing script so that it uses a numeric comparison. It tells whether an incorrect guess is high or low.

$vi prg58

```
clear
echo I\’m thinking of a number between 1 and 50.
echo Guess it and earn my approval.
read guess
until test $guess -eq 33
do
if test $guess -gt 33
then
echo Too high! Guess again.
else
echo Too low! Guess again.
fi
read guess
done
echo Well done!
```

Sample Run

$sh prg58

```
I’m thinking of a number between 1 and 50.
Guess it and earn my approval.
10
Too low! Guess again.
20
Too low! Guess again.
25
Too low! Guess again.
30
Too low! Guess again.
35
Too high! Guess again.
40
Too high! Guess again.
50
Too high! Guess again.
32
Too low! Guess again.
33
Well done!
```

59. Write a shell script that accepts the user into the Wheeler Club if his or her weight is less than 80 pounds or more than 250 pounds.

$vi prg59

```
clear
echo Greetings., What is your weight\?
read weight
if test $weight -lt 80 -o $weight -gt 250
then
```

```
echo Welcome to the Wheeler Club!
else
echo You must work to further distinguish yourself.
fi
```

Sample Run

```
$sh prg59
Greetings., What is your weight?
55
Welcome to the Wheeler Club!
$sh prg59
Greetings., What is your weight?
70
Welcome to the Wheeler Club!
$sh prg59
Greetings., What is your weight?
90
You must work to further distinguish yourself.
$sh prg59
Greetings., What is your weight?
270
Welcome to the Wheeler Club!
```

60. How will you copy a file “abc.doc” present in current directory to a directory “abc2” present in the parent directory?

Steps-

```
$mkdir abc1
$mkdir abc2
$cd abc1
$touch abc.doc
$cp abc.doc ../abc2
```

To check file is copied or not

```
$cd $ls abc
```

Output is

```
abc.doc
```

61. Write a shell script to search a file in current directory.

```
$vi prg61
clear
echo Enter a file name to search
read fn
ls | grep $fn>/dev/null
if [$? -eq 0]
then
echo The file $fn is present in the current directory.
else
echo The file $fn is not present in the current directory.
fi
```

Sample Run

```
$sh prg61
Enter a file name to search
abc.doc
The file abc.doc is not present in the current directory.
$sh prg61
Enter a file name to search
a
The file a is present in the current directory.
```

62. Write a shell script to display a three digit number in English words.

```
$vi prg62
clear
echo Enter the three digit Number
read num
a=‘expr $num % 10’
b=‘expr $num / 10’
c=‘expr $b % 10’
d=‘expr $b / 10’
set $d $c $a
for arg in $*
do
case $arg in
1) echo One ;;
2) echo Two ;;
3) echo Three ;;
4) echo Four ;;
5) echo Five ;;
6) echo Six ;;
7) echo Seven ;
8) echo Eight ;;
9) echo Nine ;;
0) echo Zero ;;
esac
done
```

Sample Run

```
$sh prg62
Enter the three digit Number
123
One
Two
Three
```

63. To find number of files in Present Working Directory.

```
$vi prg63
clear
echo Present Working Directory is:
pwd     # to display the present working directory
echo Number of files is:
```

```
pwd | ls |wc -l
```

Sample Run

```
$sh prg63
Present Working Directory is:
/root
Number of files is:
8
```

64. To display distance in different units.

```
$vi prg64
clear
echo Input distance in kilometers
read a
met=‘expr $a \* 1000’
cm=‘expr $met \* 100’
inch=‘echo $cm / 2.54 | bc’
feet=‘echo $inch / 12 |bc’
echo The distance in meters is $met meters
echo The distance in centimeters is $cm cm
echo The distance in inches is $inch inches
echo The distance in feets is $feet feets
```

Sample Run

```
$sh prg64
Input distance in kilometers
2
The distance in meters is 2000 meters
The distance in centimeters is 200000 cm
The distance in inches is 787401 inches
The distance in feets is 65616 feets
```

65. To display date and time in different formats by using positional parameters.

```
$vi prg65
clear
#Date in desired format
set ‘date’    #Setting positional parameters through date command
echo $3 $2 $6
echo $4
echo $2 $3 $6
echo $2 $6 $3
```

Sample Run

```
$sh prg65
30 Apr 2008
16:51:58
Apr 30 2008
Apr 2008 30
```

66. Moving shell files from PWD to specified directory.

```
$vi prg66
if [ $# -lt 1 ]
then
echo Improper Usage : $0 Pathname
fi
mv *.sh $1
echo All files are moved in the $1 directory
ls $1
```

Sample Run

```
$sh prg66 abc
All files are moved in the abc directory
a.sh
b.sh
$sh prg66
Improper Usage : p1 Pathname
```

67. To print all the files and total number of files in given directory.

```
$vi prg67
clear
if [ $# -lt 1 ]
then
echo Improper Usage : $0 pathname
fi
oldifs=$ifs
ifs=/
for arg in $*
do
if [ -d $arg ]
then
cd $arg
echo Present directory
echo $arg
echo Files in the directory :
ls
echo total number of files in this directory :
echo ‘ls | wc -w’
else
if [ -f $arg ]
then
echo $arg is a file
exit
fi
fi
done
ifs=$oldifs
```

Sample Run

```
$sh prg67
Improper Usage : p1 pathname
$sh prg67 /root
```

```
96
Present directory
/root
Files in the directory :
a  aaa.c  abc2  b  c  ddd  ddd1
total files in this directory :
9
```

$sh prg67 abc

```
abc is a file
exit
Desktop  p1
```

68. To sort strings.

```
$vi prg68
clear
echo Type string 1.
cat >> srt1
echo Type string 2.
cat>> str2
echo Type string 3.
cat>> str3
echo sorted strings are
sort str1 str2 str 3
```

Sample Run

$sh prg68

```
Type string 1.
abc
Type string 2.
xyz
Type string 3.
mnop
sorted strings are
abc
mnop
xyz
```

69. To find binary equivalent of a decimal number.

```
$vi prg69
clear
echo Enter a number
read a
pow=1
sol=0
while [ $a -gt 0 ]
do
x=‘expr $a % 2’
inter=‘expr $x \* $pow’
sol=‘expr $sol + $inter’
```

```
pow=‘expr $pow \* 10’
a=‘expr $a / 2’
done
echo $sol
```

Sample Run

```
$sh prg69
enter a number
12
1100
$sh prg69
Enter a number
102
1100110
$sh prg69
Enter a number
2984
101110101000
99
```

70. To calculate simple interest.

```
$vi prg70
#Calculate a simple interest
clear
echo Enter values of Principle, Time (in yrs), and rate
read p n r
si=‘expr $p \* $n \* $r / 100’
echo Simple Interest=Rs.
$si
```

Sample Run

```
$sh prg70
Enter values of Principle, Time (in yrs), and rate
2500 3 25
Simple Interest=Rs. 1875
```

71. If the sides of a triangle are denoted by a, b and c then area of the triangle is given by
area = Square root of (s(s-a)(s-b)(s-c))
where, s = (a+b+c)/2

```
$vi prg71
clear
echo Enter sides of a triangle
read a b c
s=‘expr \( $a + $b + $c \) / 2’
area=‘expr \( $s \* \( $s - $a \) \* \( $s - $b \) \* \( $s - $c \) \)’
area=‘echo sqrt \( $area \) | bc’
echo Area of the triangle is $area
```

Sample Run

```
$sh prg71
Enter sides of a triangle
```

```
60 70 50
Area of the triangle is 1469
101
```

72. Program to display system date in format MM/DD/YY & system time in format hrs:mins:secs.

$vi prg72

```
clear
echo The current system date in required format is :
date +%D
echo The current system time in required format is :
date +%T
```

Sample Run

$sh prg72

```
The current system date in required format is :
04/05/08 //  Means 5th April 2008
The current system time in required format is :
10:26:47 //  Means 10 hrs 26 mins 47 secs
```

73. Program to say hello to the user.

$vi prg73

```
clear
echo Enter your Name
read name
echo Hello $name
```

Sample Run

$sh prg73

```
Enter your Name Charles Babbage Hello Charles Babbage Enter your Name Dennis Ritchie
Hello Dennis Ritchie
```

74. Program to display a message using switch case.

$vi prg74

```
clear
echo Enter a number between 1 and 3
read num
case $num in
1) echo You have Entered 1 ;;
2) echo You have Entered 2 ;;
3) echo You have Entered 3 ;;
*) echo Please enter some value between 1 & 3 ;;
esac
```

Sample Run

$sh prg74

```
Enter a number between 1 and 3
```

```
3
You have Entered 3
$sh prg74
Enter a number between 1 and 3
2
You have Entered 2
```

75. Write a menu driven program which has following option-(
a
) Factorial of a number
(b)
Prime or not
(c)
Odd or even
(d)
Exit

```
$vi prg75
clear
ch=y
while test $ch = ‘y’
do
echo a. Factorial
echo b. Prime or not
echo c. Odd or even
echo d. Exit
echo Enter choice
read ch
case $ch in
a) echo Enter number
read num
i=1
j=1
while test $i -le $num
do
k=‘expr $i \* $j’
i=‘expr $i + 1’
j=$k
done
echo Factorial of $num is $j ;;
b) echo Enter number
read num
i=2
while test $i -lt $num
do
k=‘expr $num % $i’
if test $k -eq 0
then
echo number is not prime
break
fi
```

```
i='expr $i + 1'
done
if test $i -eq $num
then
echo number is prime ;;
fi ;;
c) echo enter number
read num
y='expr $num % 2' if test $y -eq 0
then
echo number is even
else
echo number is odd
fi ;;
d) exit ;;
*) echo wrong choice ;;
esac
echo Do you want to continue press y/n
read $ch
done
```

Sample Run

$sh prg75

```
a. Factorial
b. Prime or not
c. Odd or even
d. Exit
Enter choice
a
Enter number
4
Factorial of 4 is 24
Do you want to continue press y/n
y
a. Factorial
b. Prime or not
c. Odd or even
d. Exit
Enter choice
b
Enter number 6
number is not prime
Do you want to continue press y/n
y
a. Factorial
b. Prime or not
c. Odd or even
d. Exit
Enter choice
b
Enter number 7
number is prime
Do you want to continue press y/n y
```

```
a. Factorial
b. Prime or not
c. Odd or even
d. Exit
Enter choice
c
enter number 5 number is odd
Do you want to continue press y
a. Factorial
b. Prime or not
c. Odd or even
d. Exit
Enter choice
c
enter number 12
number is even
Do you want to continue press y
a. Factorial
b. Prime or not
c. Odd or even
d. Exit
Enter choice
2
wrong choice
Do you want to continue press y
a. Factorial
b. Prime or not
c. Odd or even
d. Exit
Enter choice
d
y/n
y/n
y/n
```

76. Program for printing user id of user whose uid >50.

```
$vi prg76
clear
cat /etc/passwd | cut -f3 -d ‘:’>aa
uid=50
while [ $uid -le 65535 ]
# 65535 is last user id
do
grep $uid aa>>bb
uid=‘expr $uid +1’
done
sort bb #
order by first digit
```

Sample Run

```
$sh prg76
```

```
500
501
502
51
65534
68
69
74
77
81
99
```

77. Program for swapping of two numbers.

```
$vi prg77
clear
echo Enter the first number
read a
echo Enter the second number
read b
c=$a
a=$b
b=$c
echo After swapping
echo first number is $a
echo  second number is $b
```

Sample Run

```
$sh prg77
Enter the first number
5
Enter the second number
6
After swapping
first number is 6
second number is 5
```

78. Write a Program to check whether a number given by the user is zero, positive or negative.

$vi prg78

```
clear
echo Enter the Number
read x
if [ $x -gt 0 ]
then
echo x is Positive
elif  [  $x -eq 0 ]
then
echo x is a Zero
else
```

```
echo x is Negative
fi
```

Sample Run

```
$sh prg78
Enter the Number
2
x is Positive


$sh prg78
Enter the Number
0
x is a Zero


$sh prg78
Enter the Number
-3
x is Negative
```

79. Program for checking the login id & password.

```
$vi prg79
clear
echo Enter the login id
read login
echo Enter the password
read password
if [ $login = root ]
then
if [ $password = redhat ]
then
echo You entered the correct login name and password
fi
else
echo login failed
fi
```

Sample Run

```
$sh prg79
Enter the login id
root
Enter the password
unix
login failed
$sh prg79
Enter the login id
root
Enter the password
redhat
You entered the correct login name and password
$sh prg79
```

```
Enter the login id
unix
Enter the password
redhat
login failed
```

80. Program to find the sum of numbers entered through command-line.

```
$vi prg80
clear
sum=0
for i in $*
do
sum=‘expr $sum + $i’
done
echo The sum of the given numbers is $sum
```

Sample Run

```
$sh prg80 30 40
The sum of the given numbers is 70
```

81. The length & breadth of a rectangle and radius of a circle are input through the keyboard. Write a program to calculate the area & perimeter of the rectangle, and the area & circumference of the circle.

$vi prg81

```
clear
echo Enter length, breadth and radius
read length breadth radius
areaR=‘expr $length \* $breadth’
perimeterR=‘expr 2 \* \( $length + $breadth \)’
areaC=‘echo 3.14 \* $radius \* $radius |bc’
‘$areaR ‘$perimeterR ‘$areaC ‘$cirC
cirC=‘echo 2\* 3.14 \* $radius |bc’
echo ‘Area of rectangle         = ‘$areaR
echo ‘Perimeter of rectangle    = ‘$perimeterR
echo ‘Area of circle            = ‘$areaC
echo ‘Circumference of circle   = ‘$cirC
```

Sample Run

```
$sh prg81
Enter length, breadth and radius
20 5 5
Area of rectangle       = 100
Perimeter of rectangle  = 50
Area of circle          = 78.50
Circumference of circle = 31.40
```

82. If a five digit number is input through the keyboard, write a program to calculate the sum of its digits.

$vi prg82

```
clear
echo Enter any five digit number
read num
d1=‘expr $num % 10’
num=‘expr $num / 10’
d2=‘expr $num % 10’
num=‘expr $num / 10’
d3=‘expr $num % 10’
num=‘expr $num / 10’
d4=‘expr $num % 10’
num=‘expr $num / 10’
d5=‘expr $num % 10’
sum=‘expr $d1 + $d2 + $d3 + $d4 + $d5’
echo Sum of digits = $sum
```

Sample Run

$sh prg82

```
Enter any five digit number
12345
Sum of digits = 15
```

83. If cost price and selling price of an item is input through the keyboard, write a program to determine whether the seller has made profit or incurred loss. Also determine how much profit was made or loss incurred.

$vi prg83

```
clear
echo Enter cost price of the item
read cp
echo Enter selling price of the item
read sp
if [ $sp -gt $cp ]
then
echo Seller had made profit
profit=‘echo $sp - $cp | bc’
echo Profit = $profit
else
if [ $cp -gt $sp ]
then
echo Seller has incurred loss
loss=‘echo $cp - $sp | bc’
echo Loss = $loss
else
echo No profit, no loss
fi
fi
```

Sample Run

$sh prg83

```
Enter cost price of the item
1500
Enter selling price of the item
```

```
2000
Seller had made profit
Profit = 500
```

84. Write a program to calculate overtime pay of employees. Overtime is paid at the rate of Rs. 12.00 per hour for every hour worked above 40 hours. Assume that employees do not work for fractional part of an hour.

$vi prg84

```
Clear
Echo How many employees are there?
Read number
emp=1
while [ $emp -le number ]
do
echo enter working hours for employee number $emp
read hours
if [ $hours -gt 40 ]
then
otpay=‘expr \( $hours - 40 \) \* 12’ echo overtime pay = Rs.
$otpay
else
echo no overtime pay
fi
emp=‘expr $emp + 1’
done
```

Sample Run

```
$sh prg84
How many employees are there?
5
enter working hours for employee number 1
12
no overtime pay
enter working hours for employee number 2
21
no overtime pay
enter working hours for employee number 3
33
no overtime pay
enter working hours for employee number 4
45
overtime pay = Rs. 60
enter working hours for employee number 5
50
overtime pay = Rs. 120
```

85. Write a program to generate all combinations of digits 1, 2 and 3 to form different numbers using for loops.

$vi prg85

```
clear
for i in 1 2 3
do
for j in 1 2 3
do
for k in 1 2 3
do
echo $i $j $k
done
done
done
```

Sample Run

$sh prg85

```
1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
1 3 1
1 3 2
1 3 3
2 1 1
2 1 2
2 1 3
2 2 1
2 2 2
2 2 3
2 3 1
2 3 2
2 3 3
3 1 1
3 1 2
3 1 3
3 2 1
3 2 2
3 2 3
3 3 1
3 3 2
3 3 3
```

86. Write a program to check whether a given number is an Armstrong number or not, An Armstrong number is one in which the sum of cube of each of the digits equals that number.

$vi prg86

```
clear
echo Enter a Number
read n
m=$n
s=0
```

```
while [ $n -gt 0 ]
do
q=‘expr $n / 10’
r=‘expr $n - \( $q \* 10 \)’
s=‘expr $s + \( $r \* $r \* $r \)’
n=$q
done
if [ $s -eq $m ]
then
echo The Number Is Armstrong
else
echo The Number Is Not Armstrong
fi
```

```
$sh prg86
Enter a Number
153
The Number Is Armstrong
$sh prg86
Enter a Number
152
The Number Is Not Armstrong
```

87. Write a program to print out all Armstrong numbers between 1 and 500. If sum of cubes of each digit of the number is equal to the number itself, then the number is called an Armstrong number. For example, 153=(1*1*1)+(5*5*5)+(3*3*3)

```
$vi prg87
clear
i=1
echo Armstrong numbers are
while [ $i -le 500 ]
do
a=‘echo $i % 10|bc’
b=‘echo $i % 100|bc’
b=‘echo \( $b - $a \) / 10|bc’
c=‘echo $i / 100|bc’
sum=‘echo \( $a \* $a \* $a \) + \( $b \* $b \*
$b \) + \($c \* $c \* $c \)|bc’
if [ $sum -eq $i ]
then
echo $i
fi
i=‘expr $i + 1’
done
```

Sample Run

```
$sh prg87
Armstrong numbers are
1
153
370
371
```

88. Write a program for swapping of two numbers without using any third variable.

```
$vi prg88
clear
echo enter numbers a and b
read a
read b
b=‘expr $a - $b’
a=‘expr $a - $b’
b=‘expr $a + $b’
echo After Swapping
echo a = $a
echo b = $b
```

Sample Run

```
$sh prg88
enter numbers a and b
12
3
After Swapping
a = 3
b = 12
$sh prg88
enter numbers a and b
21
23
After Swapping
a = 23
b = 21
123
```

89. Program to get pid of the process.

```
$vi prg89.c
#include<stdio.h>
#include<sys/types.h>
int main()
{
int pid;
pid=getpid();
printf(“The process id of the process is %d\n”,pid);
return 0;
}
```

Compile

```
$cc -o prg89 prg89.c
```

Run

```
$./prg89 Output is
The process id of the process is 4884
```

90. Program to get pid of the parent process.

```
$vi prg90.c
#include<stdio.h>
#include<sys/types.h>
int main()
{
int ppid;
ppid=getppid();
printf("The process id of the parent process
is %d\n",ppid);
return 0;
}
```

Compile

$cc -o prg90 prg90.c

Run

$./prg90

Output is

```
The process id of the parent process is 4904
```

Parent and Child Process

Any running program is called a process. From the process we can create another process. There is a parent-child relationship between these two processes. The way to achieve this is by using a function called fork(). This function splits the running process into two processes at the point where fork is called. The first is known as parent and the new process created is known as child. Both the processes have same copy of the code after the point where fork() is called.

91. Program to show how fork() divide the process into two parts.

```
$vi prg91.c
#include<stdio.h>
#include<sys/types.h>
int main()
{
printf("Hello\n");
fork(); #fork system call is used to create child
printf("World\n");
return 0;
}
```

Compile

$cc -o prg91 prg91.c

Run

$./prg91 Output is

```
Hello World World
```

92. Program to show the existence of both child and parent processes.

```
$vi prg92.c
#include<stdio.h>
#include<sys/types.h>
```

```
int main()
{
int pid;
pid=fork(); #pid=pid of child (fork()
returns pid of child process)
if(pid==0)
{
#This part gets executed in child
printf(“I am child. The value of
variable pid is %d\n”, pid);
printf(“I am child and my process
id is %d\n”, getpid());
printf(“I am child and my parent process
id is %d\n”, getppid());
}
else
{
#This part gets executed in parent
printf(“I am parent. The value of pid
is %d\n”, pid);
printf(“I am parent and my process
id is %d\n”, getpid());
printf(“I am parent and my parent process
id is %d\n”, getppid());
}
return 0;
}
```

Compile

```
$cc -o prg92 prg92.c
```

Run

$./prg92 Output is

```
I am child. The value of variable pid is 0
I am child and my process id is 4985
I am child and my parent process id is 4984
I am parent. The value of pid is 4985
I am Parent and my process id is 4984
I am Parent and my parent process id is 4822
```

Zombie and Orphans

When we fork a new child process and the parent and the child continue to execute, there are two possibilities – either the child process ends first or the parent process ends first.

If child terminates earlier than the parent then the parent process is known as Zombie.

If parent terminates earlier than the child then the child process is known as Orphan.

93. Program to show the orphan process.

```
#include<stdio.h>
#include<sys/types.h> int main()
{
int pid;
pid=fork();
if(pid==0)
{
printf("I am child and my pid is %d\n",getpid());
printf("I am child and my ppid is %d\n",getppid());
sleep(10);
printf("\nI am child and my pid is %d"\n,getpid());
printf("I am child and my ppid is %d\n",getppid());
}
else
{
printf("I am parent and my pid is %d\n",getpid());
printf("I am parent and my ppid is %d\n",getppid());
}
}
```

Compile

$cc -o prg93 prg93.c

Run

$./prg93 Output is

```
I am child and my pidis 4943
I am child and my ppid is 4942
I am parent and my pid is 4942
I am parent and my ppid is 4868
[root@localhost ~]$
I am child and my pid is 4943

I am child and my ppid is 1
```

these two lines are display

after 10 seconds

Here parent has expired so

now child is orphan

94. Program to show the Zombie process.

```
#include<stdio.h>
#include<sys/types.h>
int main()
{
if(fork()>0)
{
sleep(20);
printf("Parent\n");
}
}
```

Compile

$cc -o prg94 prg94.c

Run

$./prg94

Output is displayed after some time

```
Parent
```

95. Program to show the division of process by fork.

```
#include<stdio.h>
#include<sys/types.h>
int main()
{
int i=0,j=0,pid;
pid=fork();
if(pid==0);
{
for(i=0;i<100;i++)
printf(“%d ? ? ?”,i);
}
else
{
for(j=0;j100;j++)
printf(“%d * * *”,j);
}
printf(“\n”);
}
```

Compile

```
$cc -o prg95 prg95.c
```

Run

```
$./prg95
```

Output is display after some time

```
0 ? ? ?1 ? ? ?2 ? ? ?3 ? ? ?4 ? ? ?5 ? ? ?6 ? ? ?7 ? ? ?8 ? ? ?9 ? ?
?10 ? ? ?11 ? ? ?12 ? ? ?13 ? ? ?14 ? ? ?15 ? ? ?16 ? ? ?17 ? ? ?18 ?
? ?19 ? ? ?20 ? ? ?21 ? ? ?22 ? ? ?23 ? ? ?24 ? ? ?25 ? ? ?26 ? ? ?27
? ? ?28 ? ? ?29 ? ? ?30 ? ? ?31 ? ? ?32 ? ? ?33 ? ? ?34 ? ? ?35 ? ? ?36
? ? ?37 ? ? ?38 ? ? ?39 ? ? ?40 ? ? ?41 ? ? ?42 ? ? ?43 ? ? ?44 ? ? ?45
? ? ?46 ? ? ?47 ? ? ?48 ? ? ?49 ? ? ?50 ? ? ?51 ? ? ?52 ? ? ?53 ? ? ?54
? ? ?55 ? ? ?56 ? ? ?57 ? ? ?58 ? ? ?59 ? ? ?60 ? ? ?61 ? ? ?62 ? ? ?63
? ? ?64 ? ? ?65 ? ? ?66 ? ? ?67 ? ? ?68 ? ? ?69 ? ? ?70 ? ? ?71 ? ? ?72
? ? ?73 ? ? ?74 ? ? ?75 ? ? ?76 ? ? ?77 ? ? ?78 ? ? ?79 ? ? ?80 ? ? ?81
? ? ?82 ? ? ?83 ? ? ?84 ? ? ?85 ? ? ?86 ? ? ?87 ? ? ?88 ? ? ?89 ? ? ?90
? ? ?91 ? ? ?92 ? ? ?93 ? ? ?94 ? ? ?95 ? ? ?96 ? ? ?97 ? ? ?98 ? ? ?99
? ? ?0 * * *1 * * *2 * * *3 * * *4 * * *5 * * *6 * * *7 * * *8 * * *9
* * *10 * * *11 * * *12 * * *13 * * *14 * * *15 * * *16 * * *17 * * *18
* * *19 * * *20 * * *21 * * *22 * * *23 * * *24 * * *25 * * *26 * * *27
* * *28 * * *29 * * *30 * * *31 * * *32 * * *33 * * *34 * * *35 * * *36
* * *37 * * *38 * * *39 * * *40 * * *41 * * *42 * * *43 * * *44 * * *45
* * *46 * * *47 * * *48 * * *49 * * *50 * * *51 * * *52 * * *53 * * *54
* * *55 * * *56 * * *57 * * *58 * * *59 * * *60 * * *61 * * *62 * * *63
* * *64 * * *65 * * *66 * * *67 * * *68 * * *69 * * *70 * * *71 * * *72
* * *73 * * *74 * * *75 * * *76 * * *77 * * *78 * * *79 * * *80 * * *81
* * *82 * * *83 * * *84 * * *85 * * *86 * * *87 * * *88 * * *89 * * *90
* * *91 * * *92 * * *93 * * *94 * * *95 * * *96 * * *97 * * *98 * * *99
* * *
```

Binary Search

Suppose that the elements of the array A are sorted in ascending order, if the elements are numbers, or dictionary order if the elements are alphanumeric in nature. The best searching algorithm, called binary search, is used to find the location of the given element.

96. Write a shell script to implement the binary search algorithm.

```
$vi prg96
clear
echo Enter size of array
read size
echo Enter elements
i=0
while [ $i -lt $size ]
do
read a[$i]
i=‘expr $i + 1’ done i=0
while [ $i -lt $size ]
do
echo “${a[$i]}”
i=‘expr $i + 1’
done
echo Enter search element read num beg=0
last=‘expr $size - 1’ found=0
while [ $found -eq 0 -a $beg -le $last ]
do
mid=‘expr \( $beg + $last \) / 2
if test ${a[$mid]} -eq $num
then
echo Element is found echo Position is $mid found=1
elif ${a[$mid]} -gt $num
then
last=‘expr $mid - 1’
else
beg=‘expr $mid + 1’
fi
done
if test $found -eq 0
then
echo element is not found
fi
```

Sample Run

```
$sh prg96
Enter size of array
7
Enter elements
3
4
5
6
7
8
9
```

```
Enter search element
5
Element is found
Position is 2
```

Sample Run

```
$sh prg96
Enter size of array
6
Enter elements
4
5
6
7
8
9
Enter search element
1
element is not found
```

97. Temperature of a city in Fahrenheit degree is input through the keyboard WAP to convert this temperature into Centigrade degrees.

Formula is
c/100=f-32/180
f=9/5*c+32

```
$vi prg97
clear
echo Enter temperature in Celsius scale :
read c
f=‘echo 9 / 5 \* $c + 32 | bc’
echo
echo Equivalent temperature in Fahrenheit =  $f
```

Sample Run

```
$sh prg97
Enter temperature in Celsius scale :
60
Equivalent temperature in Fahrenheit = 92
```

98. In a town, the percentage of men is 52. Rest all are women. The percentage of total literacy is 48. If total percentage of literate men is 35 of the total population, WAP to find the total number of illiterate men and women. The population of the town is 80,000.

```
$vi prg98
clear
a=80000
totman =‘expr \( $a \* 52 \) / 100’
totwman=‘expr $a - $totman’
totLitPeople = ‘expr \( $a \* 48 \) / 100’
litman=‘expr \( $a \* 35 \) / 100’
litwman=‘expr $totLitPeople - $litman’
```

```
ilitman=‘expr $totman - $litman’
ilitwman=‘expr $totwman - $litwman’


echo ‘total man = ‘$totman
echo ‘total woman = ‘$totwman
echo ‘literate man = ‘$litman
echo ‘literate woman = ‘$litwman
echo ‘illiterate man = ‘$ilitman
echo ‘illiterate woman = ‘$ilitwman
```

Sample Run

$sh prg98

```
total man          = 41600
total woman         = 38400
literate man       = 28000
literate woman     = 13600
illiterate man     = 13600
illiterate woman   = 24800
```

99. If the three sides of a triangle are entered through the keyboard. WAP to check whether the triangle is equilateral, isosceles, or scalene triangle.

$vi prg99

```
clear
echo Enter three sides of the triangle
read a b c
echo
if [ $a -eq $b -a $a -eq $c ]
then
echo Triangle is Equilateral
elif [ $a -eq $b -o $a -eq $c -o $b -eq $c ]
then
echo Triangle is Isosceles
elif    echo Triangle is Scalene
fi
```

Sample Run

$sh prg99

```
Enter three sides of the triangle
30 75 75
Triangle is Isosceles
Enter three sides of the triangle
60 60 60
Triangle is Equilateral
Enter three sides of the triangle
38 30 35
Triangle is Scalene
```

100. An Insurance company follows following rules to calculate premium.

(

i

) If a person’s health is excellent and the person is between 25 and 35 years of age and lives in a city and is a male then Premium is Rs. 4 per thousand and his policy amount cannot exceed Rs. 2 lakhs.

(

ii

) If a person satisfies all the above conditions except that the sex is female then the premium is Rs. 3 per thousand and her policy amount cannot exceed Rs. 1 lakh.

(

iii

) if a person’s health is poor and the person is between 25 and 35 years of age and lives in a village and is a male then the Premium is Rs. 6 per thousand and his policy cannot exceed Rs. 10,000.

(

iv

) In all other cases the person is not insured.

Write a program to output whether the person should be insured or not, his/her Premium rate and maximum amount for which he/she can be insured.

```
$vi prg100
clear
echo Enter age of the person
read age
echo Enter where he lives (city or village)?
ead liv
echo Enter gender (male or female)?
read gender
echo Enter health (poor or excellent)?
read health
echo
if [ $age -ge 25 -a $age -le 35 -a $liv = ‘city’ -a $gender = ‘male’ -a $health = excellent]
then
echo The person should be insured
echo Premium is Rs.4 per thousand
echo Policy amount cannot exceed Rs.2 lakh
elif [ $age -ge 25 -a $age -le 35 -a $liv = ‘city’ -a $gender = ‘female’ -a $health = ‘excellent’ ]
then
echo The person should be insured
echo Premium is Rs.3 per thousand
echo Policy amount cannot exceed Rs.1 lakh
elif [ $age -ge 25 -a $age -le 35 -a $liv = ‘village’ -a $gender = ‘male’ -a $health = ‘poor’]
then
echo The person should be insured
echo Premium is Rs.6 per thousand
echo Policy amount cannot exceed Rs.10,000
else
echo The person should not be insured
fi
```

Sample Run

```
$sh prg100
Enter age of the person
26
```

```
Enter where he lives (city or village)?
city
Enter gender (male or female)?
male
Enter health (poor or excellent)?
excellent
The person should be insured
Premium is Rs.4 per thousand
Policy amount cannot exceed Rs.2 lakh
$sh prg100
Enter age of the person
33
Enter where he lives (city or village)?
city
Enter gender (male or female)?
female
Enter health (poor or excellent)?
excellent
The person should be insured
Premium is Rs.3 per thousand
Policy amount cannot exceed Rs.1 lakh
$sh prg100
Enter age of the person
3
Enter where he lives (city or village)?
village
Enter gender (male or female)?
male
Enter health (poor or excellent)?
poor
The person should be insured
Premium is Rs.6 per thousand
Policy amount cannot exceed Rs.10,000
$sh prg100
Enter age of the person
24
Enter where he lives (city or village)?
village
Enter gender (male or female)?
male
Enter health (poor or excellent)?
poor
The person should not be insured
```

Table of Contents