BCA
Semester-6

USEFUL FOR COMPUTER WORLD BCA STUDENT

Shell Programming

Practical

Subject Code : CC - 311

Dr. Dhara Bhatt
PhD [Computer Science]
St. Xavier's College, Navrangpura, Ahmedabad.

Dr. Bhavik Pandya
Ph.D [Computer Science]
Navgujarat College of Computer Applications,
Ashram Road, Ahmedabad

Shri Keyur Shah
MCA [Master in Computer Application], M.Phil.
System Analyst-GTU, Formerly Coordinator- CPICA.

Prof. Parul Patel
MCA (Master in Computer Applications)
Coordinator of K.K.S. Govt. BCA College,
Khokhara, Maninagar, Ahmedabad.

Reviewer: **Prof. Manisha N. Chaudhary**
M.E. (cse)
K.K.S. Govt. BCA College,
Khokhara, Maninagar, Ahmedabad.

COMPUTER WORLD PUBLICATION
A Division of Live Education System Pvt. Ltd.

COMPUTER WORLD
43, 5th Floor, SANIDHYA Complex, Nr. M. J. Library,
Opp. Sanyas Ashram, Ashram Road, Ahmedabad-09.
Mobile : 9725019114, 9725022917, 9725020595, 9825020595
URL : www.computerworld.ind.in | e-Mail :info@computerworld.ind.in

Scanned with CamScanner

Publish By :

COMPUTER WORLD

A Division of Live Education System Pvt. Ltd.

An ISO 9001 : 2008 Certified Company

Kalpesh Patel - kalpesh@computerworld.ind.in

Prepare by : Computer World Research Department

Dr. Dhara Bhatt, Dr. Bhavik Pandya,
Shri Keyur Shah, Prof. Parul Patel, Ravindrabhai

Price : ₹ 99/-

Book Code : CEBCA126

Edition : 1st

Notice of Rights

No part of this publication may be reproduced, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Computer World except under the terms of a Computer World license agreement.

Notice of Liability

The information in this courseware title is distributed on an 'as is' basis, without warranty. While very precaution has been taken in the preparation of this course, neither the authors nor Computer World shall have any liability to my person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instruction contained in this book or by the computer software and hardware products described in it.

Disclaimer

We make a sincere effort to ensure the accuracy of the material described here in however, Computer World makes no warranty, expressed or implied, with respect of the quality, correctness, reliability, accuracy, of freedom from error of this document or the products it describes. Data used in examples and sample data files are intended to be fictional. Any resemblance to real persons or companies is entirely coincidental.

Content of Copyright : COMPUTER WORLD PUBLICATION - http://www.computerworld.ind.in

Contact : 43, 5th Floor, Sanidhya Complex, Opp. Capital Comm. Center, Ashram Road, Ahmedabad-09

Mobile : 97250 22917 | 97250 19114 | 97250 20595 | 98250 20595

URL : www.computerworld.ind.in | e-Mail : info@computerworld.ind.in, info.computerworld@gmail.com

Scanned with CamScanner

INDEX

UNIT-1 General –Purpose Utilities and Files and directories 9 to 22

1.1 General Purpose Utilities:
Cal, date, echo, bc, script, who, uname, tty, man, info, passwd, logout, wc

1.2 Linux Files and Directories :
Current Working Directory-pwd, Listing Files and Directories-ls, Matching Filenames with Patterns (wildcard characters), Simple Simple ways to create a file- touch,cat,, echo, Showing the contents of a file-cat, more, less, head, tail

UNIT-2 Additional File Management Commands 23 to 47

2.1 Additional File Management Commands:
Creating Directories-mkdir, Removing Empty Directories-rmdir, Copying Files and Directories-cp, Removing Files and Nonempty Directories-rm, Renaming Files and Directories-mv, Comparing Two Files-cmp, What is Common-comm, Converting One File to Other-diff, Piping and Redirection andtee, File and Directory Permission and Privileges chmod, Locating Files-find.

2.2 Editing Files:
Creating and Editing files using vi, picoand, emacseditors.

2.3 Basics of Shell Scripting Programming:
Creating Shell Scripts using various commands of Linux except Filters.; Interactive shell script using read and echo; Decision Statements: if then fi, if then else fi, if then else if else fi caseesac; Test command; Logical Operators; Looping statements: for loop, while loop, until loop; Break, continue command; Arithmetic in Shell script using expr; Creating Shell Scripts to perform mathematical calculations.

Scanned with CamScanner

UNIT-3 Filters 48 to 60

3.1 Filters:
Simple Filters: Paginating files –pr, Splitting a file vertically–cut, Pasting files-paste, Ordering a file-sort, Locate repeated and non-repeated lines - uniq, Translating characters–tr

3.2 Filters using regular expression:
Searching for pattern-grep, stream editior –sed

3.3 Advanced filters:
Simple awkfiltering, Comparison operators, Variables, Built invariables, Control flow, looping.

UNIT-4 Compressing ,decompressing and achieving files 61 to 68

4.1 Compressing , decompressing and achieving files:
Gzip,gunzip, Tar, Zip and unzip

4.2 Environment variables:
Environment variables, Alias, Inline command editing, Miscellaneous features, Initialization script.

4.3 Communication commands:
Finger, Talk, Mesg, Mailx, Pine, Write, Wall

Practical 69 to 116

1. Check the output of the following commands: date, ls, who, cal, ps, wc, cat, uname, pwd, mkdir, rmdir, cd, cp, rm, mv, diff, chmod, grep, sed, head, tail,cut, paste, sort, find, man
2. Write a script to find the complete path for any file.
3. Write a shell script to execute following commands
   - ⇨ Sort file abc.txt and save this sorted file inxyz.txt
   - ⇨ Give an example of : To execute commands together without affecting result of Each other.

     How to print
     "thisis a three –line
     1. Textmessage"

Scanned with CamScanner

- ⇨ Which command display version of the UNIX?
- ⇨ How would you get online help of cat command?

4. Write a shell script to execute following commands
   - ⇨ How would u display the hidden files?
   - ⇨ How delete directory with files?
   - ⇨ How would user can do interactive copying?
   - ⇨ How would user can do interactive deletion offiles?
   - ⇨ Explain two functionality of "mv" command with example?

5. Write a shell script to execute following commands
   - ⇨ Create a file called text and store name,age and address init.
   - ⇨ Display the contents of the file text on the screen.
   - ⇨ Delete the directories mydir and new dir at one-shot.
   - ⇨ Sort a numeric file?
   - ⇨ Change the permissions for the file new text to666.

6. Write shell script that accept filename and displays last modification time if file exists, otherwise display appropriate message.

7. Write a shell script to display the login names that begin with 's'.

8. Write a shell script to remove the zero sized file from the current directory

9. Write a shell script to display the name of all the executable file from the current directory.

10. Write a shell script that will display welcome message according to time

11. Write a shell script to find number of ordinary files and directory files.

12. Write a shell script that takes a filename from the command line and checks whether the file is an ordinary file or not.
    - ⇨ If it is an ordinary file then it should display the contents of the file.
    - ⇨ If it is not an ordinary file then script should display the message: " File does not exist or is not ordinary, cannot display."

13. Write a shell script that takes a filename from the user and checks whether it is a directory file or not.
    - ⇨ If it is a directory, then the script should display the contents of the directory.
    - ⇨ If it is not a directory file then script should display the message: "File is not a directory file"

14. Write a shell script that takes a filename as an argument and checks if the file exists and is executable.

Scanned with CamScanner

- ⇨ If the file is executable then the shell script should display the message: "File exists"
- ⇨ If the file does not exists and is not executable then the script should display the message: "File does not exist or is not executable."

15. Write a shell script that displays all subdirectories in current working directory.

16. Write a shell script that calculates the number of ordinary and directory files in your current working directory.

17. Write a shell script that accepts 2 filenames and checks if both exists; if both exist then append the content of the second file into the first file.

18. Write a shell script that takes the name of two files as arguments and performs the following:
    i. Displays the message :
    "Displaying the contents of file :(first argument)" and displays the contents page wise.
    ii. Copies the contents of the first argument to second argument.
    iii. Finally displays the message: "File copied successfully."

19. Write a shell script to display the following menu and acts accordingly:
    i. Calendar of the current month and year.
    ii. Display "Good Morning/Good Afternoon/Good Evening" according to the current login time.
    iii. User name, Users home directory.
    iv. Terminal name, Terminal type.
    v. Machine name.
    vi. No. of users who are currently logged in; List of users who are currently logged in.

20. Write a shell script that displays the following menu and acts accordingly
    1. Concatenates two strings
    2. Renames a file.
    3. Deletes a file.
    4. Copy the file to specific location

21. Write a shell script to change the suffix of all your *.txt files to .dat.

22. Write a shell script to accept a directory-name and display its contents. If input is not given then HOME directory's contents should be listed. (Make use of command line argument)

23. Write a shell script to get all files of home directory and rename them if their names start with c.

    Newname = oldname111

Scanned with CamScanner

24. Write a shell script that takes two filename as arguments. It should check whether the contents of two files are same or not, if they are same then second file should be deleted.

25. Write a shell script that accepts two directory names from the command line and copies all the files of one directory to another. The script should do the following:
    - If the source directory does not exist, flash a error message
    - If destination directory does not exist create it
    - Once both exist copy all the files from source directory to destination directory.

26. Write a shell script that displays the following menu
    - List home directory
    - Date
    - Print working directory
    - Users logged in

Read the proper choice. Execute corresponding command. Check for invalid choice.

27. Write a shell script that displays all hidden files in current directory.

28. Write a shell script that Combine two files in the third file horizontally and vertically.

29. Write a shell script to delete all the spaces from a given file.

30. Write a shell script to find a given date fall on a weekday or a weekend.

31. Write a shell script to search for a given word in all the files given as the Arguments on the command line.

32. Write a shell script that display last modified file in the current directory.

33. Write a script to display the permissions of the particular file.

34. Write a shell script to display the calendar in the following manner:
    i. Display the calendar of months m1 and m2 by 'CAL m1, m2' command file.
    ii. Display the calendar of the months from m1 to m2 by 'CAL m1-m2' command file.

35. Write a shell script to display the following menu for a particular file :
    i. Display all the words of a file in ascending order.
    ii. Display a file in descending order.
    iii. Toggle all the characters in the file.
    iv. Display type of the file.

36. Write a shell script to check whether the named user is currently logged in or not.
37. Write a shell script to display the following menu for a particular file:

Scanned with CamScanner

i. Display all the words of a file in ascending order.
ii. Display a file in descending order.
iii. Display a file in reverse order.
iv. Toggle all the characters in the file
v. Display type of the file.

38. Write a shell script to find total no. Of users and finds out how many of them are currently logged in.

39. Write a shell script that displays the directory information in the following format
Filename Size Date Protection Owner

40. Write a shell script to display five largest files from the current directory.

41. Write a shell script that toggles contents of the file

42. Write a shell script that report whether your friend has currently logged in or not. If he has logged in then the shell script should send a message to his terminal suggesting a dinner tonight. If you do have write permission to his terminal or if he hasn't logged in then such a message should be mailed to him about your dinner proposal.

43. Write a shell script for the performing the write and mail.

44. Write a shell script to accept any character using command line and list all the files starting with that character in the current directory.

45. Create a file called student containing roll-no, name and marks.
a. Display the contents of the file sorted by marks in descending order
b. Display the names of students in alphabetical order ignoring the case.
c. Display students according to their rollnos.
d. Sort file according to the second field and save it to file'names'.
e. Display the list of students who scored between 70 and80.ize Date Protection Owner

Scanned with CamScanner

UNIT-1

General –Purpose Utilities and Files and directories

- General Purpose Utilities:

  Cal, date, echo, bc, script, who, uname, tty, man, info, passwd, logout, wc.

- Linux Files and Directories:

  Current Working Directory-pwd, Listing Files and Directories-ls, Matching Filenames with Patterns (wildcard characters), Simple ways to create a file- touch, cat,, echo, Showing the contents of a file-cat, more, less, head, tail

Scanned with CamScanner

Unit -1 General -Purpose Utilities and Files and Directories Introduction

1.1 General Purpose Utilities:

Unix/Linux general purpose utility command list: man, who, cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, finger, pwd, cal, logout, shutdown. From that some of general-purpose utilities command we will discuss: Cal, date, echo, bc, script, who, uname, tty, man, info, passwd, logout, wc.

1. **Calendar**: cal command. Its use to display the calendar.

Syntax:

cal [options] [month] [year]

Examples:

⇨ cal

In general, if no options are given, cal displays the current month at the command line.

⇨ cal 12 2000

Displays the calendar for December of the year 2000.

⇨ cal -3 5 2008

Displays the calendar of April, May and June month of the year 2008.

```
[root@localhost ~]#
[root@localhost ~]# cal
    January 2014
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
[root@localhost ~]# cal 12 2000
     December 2000
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
[root@localhost ~]# cal -3 5 2008
      April 2008                          April 2008                          April 2008
Su Mo Tu We Th Fr Sa     Su Mo Tu We Th Fr Sa     Su Mo Tu We Th Fr Sa
          1  2  3  4  5                    1  2  3     1  2  3  4  5  6  7
 6  7  8  9 10 11 12     4  5  6  7  8  9 10     8  9 10 11 12 13 14
13 14 15 16 17 18 19    11 12 13 14 15 16 17    15 16 17 18 19 20 21
20 21 22 23 24 25 26    18 19 20 21 22 23 24    22 23 24 25 26 27 28
27 28 29 30             25 26 27 28 29 30 31    29 30
```


Scanned with CamScanner

- **Calendar other options:**

Options	Meaning
-1	Displays the single month as output.
-3	Displays prev/current/next month output.
-s	Displays Sunday as the first day of the week.
-m	Displays Monday as the first day of the week.
-j	Displays Julian dates (days one-based, numbered from January 1).
-y	Displays a calendar for the current year.

2. **Date:** You can display the current date with date command, which shows the date and time to the nearest second:

Date: The date command can also be used with suitable format specifier as arguments.

You can print only the month using the format +%m

date +%m

Or month name

date +%h

Combining both the options.

date +"%h %m"

There are many other format specifiers, and useful ones are listed below :

d – The day of the month (1 to 31)

y – The last two digits of the year

H, M, and S – The hour, minute and second, respectively.

T – The time in the format hh:mm:ss

When you use multiple format specifier, you must enclose them within quotes (single or double), and use a single + symbol before it.

Scanned with CamScanner

```
[root@localhost  ~]#
[root@localhost  ~]#
[root@localhost  ~]#  date
Tue   Jan  28   17:49:36 EST 2014
[root@localhost  ~]#
[root@localhost  ~]#
[root@localhost  ~]# date +%m
01
[root@localhost  ~]#
[root@localjost  ~]#  date +%h
Jan
[root@localhost  ~]#  date +" %m  %h"
01  Jan
[root@localhost  ~]#  date +%d
28
[root@localhost  ~]#  date +%y
14
[root@localhost  ~]#
[root@localhost  ~]#  date +" %H   :   %M    : % S "
 17 : 51 : 39
[root@localhost  ~]#  date +%T
17:51:50
[root@localjost  ~]#
```

3. **Echo:** display a line of text.

Syntax: echo [option(s)] [string(s)]

A string is any finite sequence of characters (i.e., letters, numerals, symbols and punctuation marks). echo This is a pen. Displaying the value of variable x. echo The number is Sx. echo, by default, follows any output with a newline character. This is a non-printing (i.e., invisible) character that represents the end of one line of text and the start of the next

Scanned with CamScanner

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# echo –e "\n Project1: \n\n\tplan \n\tweite\n\ttest\n"> project1
[root@localhost ~]# cat project1

Project1:

    plan
    write
    test

[root@localhost ~]# echo –E "\n Projects: \n\n\tplan \n\tcode \n\ttest\n">
\n Projects: \n\n\tplan \n\tcode \n\ttest\n
[root@localhost ~]#
[root@localhost ~]#
```

4. **bc:** The calculator

bc is a text-based calculator. When you invoke bc without arguments, the cursor keeps on blinking and nothing seems to happen. bc belongs to a family of commands (called filters) that expects input from keyboard when used without an argument. Use [ctrl-d] to quit bc

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# bc
bc 1.06.95
Copyright 19991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.

123*2
246
412/2
206
230+400-12/2
624
(23 + 45) – (43*3) * 9 – (456/2)
-1321
exit
0
quit
[root@localhost ~]#
```


Scanned with CamScanner

5. script: Recording your session

script command lets you record your login session in a file. You can later view the file. If you are doing some important work and wish to keep a log of all your activities, you should invoke this command immediately after you login:

script

Script started, the file is typescript

exit

Script done; the file is typescript. You can now view this file with the cat command. Script overwrites any previous typescript that may exist. If you want to append to it, or want to use a different log file, you can consider below arguments.

```
[root@localhost ~]#
[root@localhost ~]# script logfile
Script started. File is logfile
[root@localhost ~]# echo "My shell is $SHELL"
My shell is /bin/bash
[root@localhost ~]#
[root@localhost ~]# printf "My shell is $SHELL"
My shell is /bin/bash [root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# pwd
/root
[root@localhost ~]# echo $USER
Root
[root@localhost ~]# exit
exit
Script done, file is logfile
[root@localhost ~]# cat logfile
Script started on Tue 28 Jan 2014 05:59:54 PM EST
[root@localhost ~]# echo "My shell is $SHELL"
My shell is /bin/bash
[root@localhost ~]#
[root@localhost ~]# printf "My shell is $SHELL"
My shell is /bin/bash [root@localhost ~] #
[root@localhost ~]#
[root@localhost ~]# pwd
/root
[root@localhost ~]# echo $USER
Root
[root@localhost ~]# exit
exit
Script started on Tue 28 Jan 2014 06:01:13 PM EST
[root@localhost ~]#
```


Scanned with CamScanner

6. **who:** Linux maintains an account of all users who are logged on to the system. It's often a good idea to know their user-ids so you can mail them messages.

First column Shows usernames (or user-ids)

Second column The second column shows device names of their respective terminals

The third column, Fourth column & Fifth column,Shows the date and time of logging in.

Sixth column Shows machine name from where the user logged in

who –Hu

-H for header option –u for more detailed list.

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# who
root     :0       2014-01-28  14:51 (:0)
root     pts/0    2014-01-28  14:52 (:0)
[root@localhost ~]#
[root@localhost ~]# who –Hu
Name          Line     Time                IDLE        PID COMMENT
root          :0       2014-01-28  14:51   ?           1408 (:0)
root          pts/0    2014-01-28  14:52   old         1793 (:0)
root@localhost ~]#
[root@localhost ~]# who am i
root     pts/0         2014-01-28  14:52 (:0)
[root@localhost ~]#
[root@localhost ~]#
```

7. **uname:** Knowing your machine's characteristics

The uname command displays certain features of the operating system running on your machine.To simply display the name of operating system

uname

Use –r option to know the version of your operating system

uname -r

If your machine is connected to a network, it must have a name (called hostname).

uname –n

The same output is obtained with hostname command:

Scanned with CamScanner

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# uname
Linux
[root@localhost ~]#
[root@localhost ~]# uname -r
3.6.10-4. fc18. *86_64
[root@localhost ~]#
[root@localhost ~]# uname -n
Localhost.localdomain
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
```

8. **tty:** Know your Terminal

Since Unix/Linux treats even terminals as files, it's reasonable to expect a command that tells you the filename of the terminal you are using. It's the tty (teletype) command. This command is simple and need no arguments.

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# tty
/dev/pts/0
[root@localhost ~]#
[root@localhost ~]#


[root@localhost ~]#
```

9. **man:** man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO.

Every manual is divided into the following sections:

- Executable programs or shell commands

Scanned with CamScanner

- System calls (functions provided by the kernel)
- Library calls (functions within program libraries
- Games
- Special files (usually found in /dev)
- File formats and conventions eg /etc/passwd
- Miscellaneous (including macro packages and conventions), e.g. groff(7)
- System administration commands (usually only for root)
- Kernel routines [Non standard]

Syntax :

$man [OPTION]... [COMMAND NAME]...

10. info: info command reads documentation in the info format. It will give detailed information for a command when compared with the man page. The pages are made using the texinfo tools because of which it can link with other pages, create menus and easy navigation.

Syntax:

info [OPTION]... [MENU-ITEM...]

Options:

-a, –all: It use all matching manuals.

-k, –apropos=STRING: It look up STRING in all indices of all manuals.

-d, –directory=DIR: It add DIR to INFOPATH.

-f, –file=MANUAL: It specify Info manual to visit.

-h, –help: It display this help and exit.

-n, –node=NODENAME: It specify nodes in first visited Info file.

-o, –output=FILE: It output selected nodes to FILE.

-O, –show-options, –usage: It go to command-line options node.

-v, –variable VAR=VALUE: It assign VALUE to Info variable VAR.

–version: It display version information and exit.

-w, –where, –location: It print physical location of Info file.

Examples:

-a : It use all matching manuals and display them for a particular command.

info -a cvs

Scanned with CamScanner

11. passwd: passwd command in Linux is used to change the user account passwords. The root user reserves the privilege to change the password for any user on the system, while a normal user can only change the account password for his or her own account.

Syntax:

passwd [options] [username]

12. logout: session-logout.

logout command allows you to programmatically logout from your session. causes the session manager to take the requested action immediately.

Syntax:

session-logout[options]

13. wc: Counting Lines, Words & Characters

wc is a word-counting program that counts lines, word and characters. It takes one or more filename as arguments and displays a four-columnar output.

Consider a regular text file sample with the following contents:

I am the wc command

I count lines, words, and characters

With options I can also make a selective count

```
[root@localhost ~]#
[root@localhost ~]# cat sample
I am the wc command
I count lines, words and characters
With options I can also make a selective count
[root@localhost ~]#
[root@localhost ~]# wc sample
  3   20  103 sample
[root@localhost ~]#
[root@localhost ~]# wc – l sample
3 sample
[root@localhost ~]# wc – w sample
20 sample
[root@localhost ~]# wc – c sample
103 sample
[root@localhost ~]#
```


Scanned with CamScanner

1.2 Linux Files and Directories:

Current Working Directory: The current working directory is the directory in which the user is currently working in. Each time you interact with your command prompt, you are working within a directory. By default, when you log into your Linux system, your current working directory is set to your home directory. To change the working directory use the cd command. For example, to change the current working directory to /tmp you would type:

cd /tmp

a) **pwd:** The pwd command stands for print working directory. It is one of the most basic and frequently used commands in Linux. When invoked the command prints the complete path of the current working directory.

pwd is a shell builtin in most modern shells such as bash and zsh. Its behavior is slightly different than the standalone /bin/pwd executable. You can use the type command to display all locations containing pwd:

type -a pwd

output:

pwd is a shell builtin

pwd is /bin/pwd

As you can see from the output below, the shell builtin has priority over the standalone executable and it is used whenever you type pwd. If you want to use the standalone pwd binary type the full path to the file /bin/pwd

b) **ls:** Listing Files and Directories

ls is one of the basic commands that any Linux user should know. The ls command lists files and directories within the file system, and shows detailed information about them. It is a part of the GNU core utilities package which is installed on all Linux distributions. This article will show you how to use the ls command through practical examples and detailed explanations of the most common ls options.

Syntax:

ls [OPTIONS] [FILES]

When used with no options and arguments, ls displays a list of the names of all files in the current working directory :

$ ls

The files are listed in alphabetical order in as many columns as can fit across your terminal:

Output:

Scanned with CamScanner

cache db empty games lib local lock log mail opt run spool tmp

c) **Wildcards and Pattern Matching**: Wildcards can be used in two different ways. They can be used to specify a single location or file by using a wildcard to represent a character or characters, or they can be used to reference multiple files with a single command.

These wildcards are interpreted by the shell and the results are returned to the command you run. There are three main wildcards in Linux:

- ⇨ An asterisk (*) – matches one or more occurrences of any character, including no character.
- ⇨ Question mark (?) – represents or matches a single occurrence of any character.
- ⇨ Bracketed characters ([ ]) – matches any occurrence of character enclosed in the square brackets. It is possible to use different types of characters (alphanumeric characters): numbers, letters, other special characters etc.

There are several wildcards that can be used in the bash shell. The most common wildcards are * and - . The wildcard * can represent 0, 1 or more of any string of regular characters. The wildcard - represents exactly 1 of any character.

In this use of a wildcard, a single location is being specified.

$ cd /home/je*/notes

Since there is only one directory in /home that starts with je, the wildcard is translated to fill in the rest of the directory name and the cd command succeeds.

In this example, use the same wildcard to specify multiple files which match a pattern.

$ ls chapter*

chapter1.txt chapter3.txt chapter5.txt

chapter2.txt chapter4.txt

There are multiple matches, but the ls command is capable of handling multiple files, so the command again succeeds.

d) **Simple ways to create a file:** To create a file there are three commands. Touch, cat and echo. Below is given description and their examples of each command.

- **touch** command with any extension to create file, this command will create an empty file touch.txt in your current directory as an example below.

  $ sudo touch touch.txt

  $ sudo touch touch.docx

- **cat** command to create file, this command will create an empty file cat.txt in your current directory as an example below, but you must add text in the file.

Scanned with CamScanner

$ cat > cat.txt

Add the text below.

This file has been created with cat command

To save the file hit Ctrl + d, and to see the file type command below.

- **echo** command to create file, this command will create a file echo.txt in your current directory as an example below, but you should add text in the line command.

  $ echo "This file has been created with echo command" > echo.txt

e) Showing the contents of a file-cat, more, less, head, tail:

- **cat:** can be used to join multiple files together and print the result on screen (it will not show page by page)

  Example:

  cat 01.txt

  to display the contents of file 01.txt

  cat 01.txt 02.txt

  to display the contents of both files

  cat file1.txt file2.txt > file3.txt

  Reads file1.txt and file2.txt and combines those files to make

  cat note5 >> notes – attach note5 to notes

  cat >> file1 – add additional data in file1

- **more:** to view a text file one page at a time, press spacebar to go to the next page

  more filename : show the document one page at a time

  more -num filename : show the document page few lines as specified bu (-num)

  example : more -10 filename will show 10 lines for every page

- **less:** is much the same as more command except:
  i. You can navigate the page up/down using the less command and not possible in more command.
  ii. You can search a string in less command. (use /keywordto search)
  iii. "more" was fairly limited, and additional development on "more" had stopped
  iv. it uses same functions as vi editor

  Example: less filename

- **head:** displays the first ten lines of a file, unless otherwise stated.

  Examples:

Scanned with CamScanner

head myfile.txt – Would display the first ten lines of myfile.txt.

head -15 myfile.txt – Would display the first fifteen lines of myfile.txt.

- **tail:** display the last part of the file

  Examples: tail filename

  tail -n filename : display the last n lines of the file

☺ ☺ ☺ ☺ ☺

Scanned with CamScanner

UNIT-2

Additional File Management Command

- **Additional File Management Commands:**

  Creating Directories-mkdir, Removing Empty Directories-rmdir, Copying Files and Directories-cp, Removing Files and Nonempty Directories-rm, Renaming Files and Directories-mv, Comparing Two Files-cmp, What is Common-comm, Converting One File to Other-diff, Piping and Redirection andtee, File and Directory Permission and Privileges
  chmod, Locating Files-find.

- **EditingFiles:**

  Creating and Editing files using vi, picoand, emacseditors.

- **Basics of Shell Scripting Programming:**

  Creating Shell Scripts using various commands of Linux except Filters.; Interactive shell script using read and echo; Decision Statements: if then fi, if then elsefi, if then elif elsefi, caseesac; Test command; Logical Operators; Looping statements: for loop, while loop, until loop; Break, continue command; Arithmetic in Shell script using expr; Creating Shell Scripts to perform mathematical calculations.

Scanned with CamScanner

Unit –2 Additional File Management Command

2.1 Additional File Management Command:

➢ Creating Directories - mkdir command:

mkdir command in Linux allows the user to create directories (also referred to as folders in some operating systems). This command can create multiple directories at once as well as set the permissions for the directories. *mkdir* stands for "make directory."

Syntax:

mkdir [options...] [directories ...]

Option / Syntax	Description
mkdir directory_name	Creates a directory in the current location
mkdir {dir1,dir2,dir3,dir4}	Creates multiple directories in the current location. Do not use spaces inside {}
mkdir –p directory/path/newdir	Creates a directory structure with the missing parent directories (if any)
mkdir –m777 directory_name	Creates a directory and sets full read, write, execute permissions for all users
mkdir –v directory_name(s)	Creates a directory in the current location and give feedback for successful operations. It is also called verification.

Examples:

Sr. No.	Command	Description / Output
1.	mkdir bca	Create bca directory.
2.	mkdir {sem1,sem2,sem3}	Create Multiple directories like sem1, sem2 and sem3.
3.	mkdir –p bca/bca6/linux	It creates bca6 and linux directory in bca directory.

Scanned with CamScanner

4.	mkdir –m777 gujarat	It creates gujarat directory with all the permissions to all users.
5.	mkdir –v gujarat/ahm	It creates ahm directory in gujarat directory.

- **Remove Directories with rmdir Command:**

rmdir command is used only when deleting empty folders and directories in Linux. If a specified directory is not empty, the output displays an error.

The basic syntax used for **removing empty Linux folders/directories** is:

Syntax:

rmdir [dir_name]

Additionally, you can **delete multiple empty directories** at once by typing:

Syntax:

rmdir [dir_name1][dir_name2][dir_name3]

If the command finds content in one of the listed directories, it will skip it and move on to the next one.

Example:

Sr. No.	Command	Description / Output
1.	rmdir gujarat	Remove gujarat directory
2.	rmdir sem1 sem2 sem3	Removes sem1, sem2 and sem3 directory.

- **Remove Directory with rm Command:**

rm command removes a directory/folder along with all the files and sub-directories in it.

By adding the -r (-R) option to the rm command, you can remove a directory along with all its contents.

To **remove a directory** (and everything inside of it) use the –r option as in the command:

Syntax:

rm –r dir_name

This will prompt you for confirmation before deleting.

To **remove a directory without confirmation**:

rm –rf directory

Also, you can **delete more than one directory** or folder at a time:

rm –r dir_name1 dir_name2 dir_name3

Scanned with CamScanner

Example:

Sr. No.	Command	Description / Output
1.	rmdir –r gujarat	Remove gujarat directory with all subdirectories and files.
2.	rmdir –r sem1 sem2 sem3	Removes sem1, sem2 and sem3 directory with subdirectories and files.

➢ **Copying Files and Directories:**

The cp command is the primary method for copying files and directories in Linux.

Syntax:

cp [additional_option] source_file target_file

Example:

cp my_file.txt my_file2.txt

This Linux command creates a copy of the *my_file.txt* file and renames the new file to ***my_file2.txt.***

By default, the cp command runs in the same directory you are working in. However, the same file cannot exist twice in the same directory.

⇨ **Copy File into another directory:**

To copy a file from the directory you're working in to a different location, use the command:

Syntax:

cp my_file.txt /new_directory

You don't need to rename the file unless there's already one with the same name in the target directory.

To specify a path for the source file:

cp /etc/my_file.txt /new_directory

This lets you copy without having to change directories. The cp command will create the **/new_directory** if it doesn't exist.

To rename and copy a file to a different path:

cp my_file.txt /new_directory/my_file2.txt

Example:

Scanned with CamScanner

Sr. No.	Command	Description / Output
1.	cp myfile.txt myfile2.txt	Copy myfile.txt file into the same directory as a myfile2.txt
2.	cp myfile.txt /bca	Copy myfile.txt file into the bca directory as a myfile.txt
3.	cp myfile.txt /bca/myfile2.txt	Copy myfile.txt file into the bca directory as a myfile2.txt
4.	cp myfile.txt /bca/myfile2.txt	Copy myfile.txt file into the bca directory as a myfile2.txt

- **Renaming Files and Directories-mv:**

mv command is a command that is used primarily to move files and folder from one location to another. However, it can also be used to rename a file. It is also used to rename the directory.

Syntax:

mv (option) filename1 filename2

Example:

Sr. No.	Command	Description / Output
1.	mv ahmedabad.txt ahm.txt	Rename ahmedabad.txt file to ahm.txt
2.	mv –v ahmedabad.txt ahm.txt	Rename ahmedabad.txt file to ahm.txt as well as it display output (verbose) on your screen. Output: renamed ahmedabad.txt -> ahm.txt
3.	mv dir_india dir_hindustan	Rename dir_india directory to dir_hindustan.
4.	mv /home/baroda.txt /home/gujarat/bar.txt	Rename baroda.txt file to bar.txt and move this file into gujarat directory.

- **Comparing Two Files:**

cmp command in Linux/UNIX is used to compare the two files byte by byte and helps you to find out whether the two files are identical or not.

Scanned with CamScanner

- When cmp is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and if no difference is found *i.e* the files compared are identical.
- cmp displays no message and simply returns the prompt if the files compared are identical.

Syntax:

cmp (OPTION) FILE1 FILE2

Example:

Sr. No.	Command	Description / Output
	Suppose myfile.txt contains Hello How are you. ***Suppose myfile2.txt contains*** Hello I am fine.	
1.	cmp myfile.txt myfile2.txt	It compares both the file contents identically. If it is not same then it displays byte and line number. If it is same then no message will be displayed. Output: myfile.txt myfile2.txt differ: byte 7, line 1
2.	cmp -b myfile.txt myfile2.txt	It displays the differing bytes in the output when used with -b option Output: myfile.txt myfile2.txt differ: byte 7, line 1 is 110 H 111 I
3.	cmp -l file1.txt file2.txt	-l option displays the byte position and byte value for all differing bytes. 7 110 111 8 157 40 9 167 141 10 40 155 11 141 40 12 162 146

Scanned with CamScanner

		13 145 151 14 40 156 15 171 145 16 157 56 17 165 12 In our example, it differs from byte 7 so it displays 7 then ascii value of (H)-110 from myfile.txt and ascii value of (I)-111 then continue for all other characters.
4.	cmp -i 7:17 myfile.txt myfile2.txt	-i option skip the 7th to 17th bytes from both the file so remaining characters are same (Hello) so it displays nothing (no message) means contents are matched.

➢ **Converting One File to Other – diff:**

diff stands for **difference**. This command is used to display the differences in the files by comparing the files line by line. This command displays which lines in one file is required to be changed to make the two files identical.

Special symbols are:

a : add

c : change

d : delete

Syntax:

diff (OPTION) FILE1 FILE2

The first line of the diff output contains:

- line numbers corresponding to the first file,
- a letter (a for *add*, c for *change*, or d for *delete*), and
- line numbers corresponding to the second file.

In our output above, "2,4c2,4" means: "Lines 2 through 4 in the first file need to be changed to match lines 2 through 4 in the second file." It then tells us what those lines are in each file:

- Lines preceded by a < are lines from the first file;
- lines preceded by > are lines from the second file.

The three dashes ("---") merely separate the lines of file 1 and file 2.

Scanned with CamScanner

Example:

Suppose states.txt contains	*Suppose otherstates.txt contains*	*Suppose newstates.txt contains*	*Suppose latest.txt contains*
Gujarat Rajasthan	Gujarat Rajasthan	Gujarat Rajasthan Uttarakhand Jharkhand	Gujarat Rajasthan Kerala Uttarakhand Jharkhand

Sr. No.	Command	Description / Output
1.	diff states.txt otherstates.txt	It compares both the file contents identically. Contents of both the files are same so *no message* will be displayed.
2.	diff states.txt newstates.txt	It compares both the file contents identically. Contents of both the files are different so *it will show line number*. Output: 1,2c1,4 < Gujarat < Rajasthan --- > Gujarat > Rajasthan > Uttarakhand > Jharkhand
	In our output above, "1,2c1,4" means: "Lines 1 through 2 in the first file need to be changed to match lines 1 through 4 in the second file." It then tells us what those lines are in each file: • Lines preceded by < are lines from the first file; • Lines preceded by > are lines from the second file. The three dashes ("---") merely separate the lines of file 1 and file	
3.	diff newstates.txt latest.txt	Output: 2a3 > Kerala

Scanned with CamScanner

	Here, 2*a*3 means requires to add Kerala in 3rd line of newstates.txt file to match the contents with latest.txt file.	
4.	diff latest.txt newstates.txt	3d2 < Kerala
	Here, 3*d*2 means requires to delete 3rd line (Kerala) from latest.txt file to match the contents with newstates.txt file.	

- **Piping – Redirection:**

Pipe is used to combine two or more commands. Redirection means to transfer of standard output to some other destination and for that pipe character '|' is used.

Syntax :

command_1 | command_2 | command_3 | .... | command_N

Example:

Suppose

latest.txt contains

Gujarat

Rajasthan

Kerala

Uttarakhand

Jharkhand

Sr. No.	Command	Description / Output
1.	ls – l | more	The more command takes the output of $ ls -l as its input. The net effect of this command is that the output of ls -l is displayed one screen at a time.
2.	ls -l > temp.txt	It stores the output of ls –l in temp file.
3.	cat latest.txt | head -3	Output: Gujarat Rajasthan Kerala
4.	cat latest.txt | tail -2	Output: Uttarakhand

Scanned with CamScanner

		Jharkhand
5.	cat latest.txt | head -2 > temp.txt	Following output is stored in temp.txt file. Gujarat Rajasthan

➢ **Tee:**

The tee command is used to reads the standard input and writes it to one or more files, along with the standard output. The tee command is commonly used along with other commands with pipe (|).

Syntax :

tee (options) FILES…

Example:

Suppose

myfile.txt contains

Gujarat

Rajasthan

Gujarat

Uttarakhand

Gujarat

Sr. No.	Command	Description / Output
1.	echo "Hello How are you" | tee file1.txt	Output: Hello How are you It creates file1.txt and store Hello How are you.
2.	echo "Hello How are you" | tee file1.txt echo "_____" ls -l | grep file1 echo "_____" cat file1.txt	Output: Hello How are you _____ -rwxrwxrwx 1 root root 18 Dec 24 07:37 file1.txt _____ Hello How are you ********** It creates file1.txt and store Hello How are you.

Scanned with CamScanner

3.	cat states.txt | grep "Gujarat" | tee file2.txt | wc –l	Output: 3 It creates file2.txt and stores Gujarat 3 times.

➢ **File and Directory Permission and Privileges – chmod:**

chmod command is used to change the access mode of a file. The name is an abbreviation of change mode. In Linux, each file is associated with an owner and a group and assigned with permission access rights for three different classes of users:

- The file owner
- The group members
- Others (everybody else)

There are three file permissions types that apply to each class. Each write, read, and execute permissions have number values which is another option to set the permissions.

- The read permission - Number value is 4
- The write permission - Number value is 2
- The execute permission - Number value is 1

chmod command allows you to specify which users are allowed to read the file, write to the file, or execute the file. The operator is used to specify how the modes of a file should be adjusted. The following operators are accepted:

Operator	Description
+	Adds the specified modes to the specified classes.
-	Removes the specified modes from the specified classes.
=	The modes specified are to be made the exact modes for the specified classes

Syntax:

chmod [reference][operator][mode] file...

Example:

Sr. No.	Command	Description / Output
1.	ls –l	It will display files and directories with permissions.
	-rwxrwxrwx 1 root root 21 Dec 28 12:03 gujarat.txt Here, - means it is file not directory and rwx*rwx*rwx means read, write and execution permission to user, *group* and others.	

Scanned with CamScanner

2.	chmod 642 gujarat.txt ls –l	It will change the permission of gujarat.txt file as under User – Read & Write (4+2) Group – Read (4) Others – Write (2) -rw- r-- -w- 1 root root 21 Dec 28 12:04 gujarat.txt
3.	chmod u=rwx gujarat.txt	It gives user to read, write and execute permissions.
4.	chmod u+x gujarat.txt	It adds execute permissions to user.
5.	chmod u-r gujarat.txt	It removes the read permission from user.
6.	chmod u=rwx,g=r,o= filename	It gives read, write and execute permission to user, read permission to group and no permission to others.
7.	chmod 777 gujarat.txt	It gives read, write and execute permissions to user, group and others.

⇨ ***How to change the permission to directory recursively:***

To recursively operate on all files and directories under the given directory, use the –R option.

Example: chmod –R 755 /root/India

It will change the permission of all the files and subdirectories of the India directory recursively.

Locating Files using Find command:

The find command in Linux is a command line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the '-exec' other UNIX commands can be executed on files or folders found.

Syntax:

find [where to start searching from] [expression determines what to find] [-options] [what to find]

Scanned with CamScanner

Examples:

1. Search a file with specific name.

find ./india -name gujarat.txt

It will search for gujarat.txt in india directory.

2. Search a file with pattern.

find ./india -name *.txt

It will give all files which have '.txt' at the end.

3. How to find and delete a file with confirmation.

find ./india -name gujarat.txt -exec rm -i {} \;

When this command is entered, a prompt will come for confirmation, if you want to delete gujarat.txt or not. if you enter 'Y/y' it will delete the file.

4. Search for empty files and directories.

find ./india -empty

This command find all empty folders and files in the entered directory or sub-directories.

5. Search for file with entered permissions.

find ./india -perm 664

This command find all the files in the india directory or sub-directory with the given permissions.

6. Search text within multiple files.

find ./ -type f -name "*.txt" -exec grep 'hello' {} \;

This command print lines which have 'hello' in them and '-type f' specifies the input type is a file.

2.2 Editing Files:

Editing Files using vi, pico and emacs editors:

There are two types of text editors in Linux:

1. Text Editors: are used to run in terminal windows.
2. Graphical Editors: are used to run in graphical windows with menus and mouse pointers.

❖ Text Mode Editors:

The three text mode editors of choice in LINUX are vi, emacs, and pico.

Scanned with CamScanner

➢ **vi Editor:**

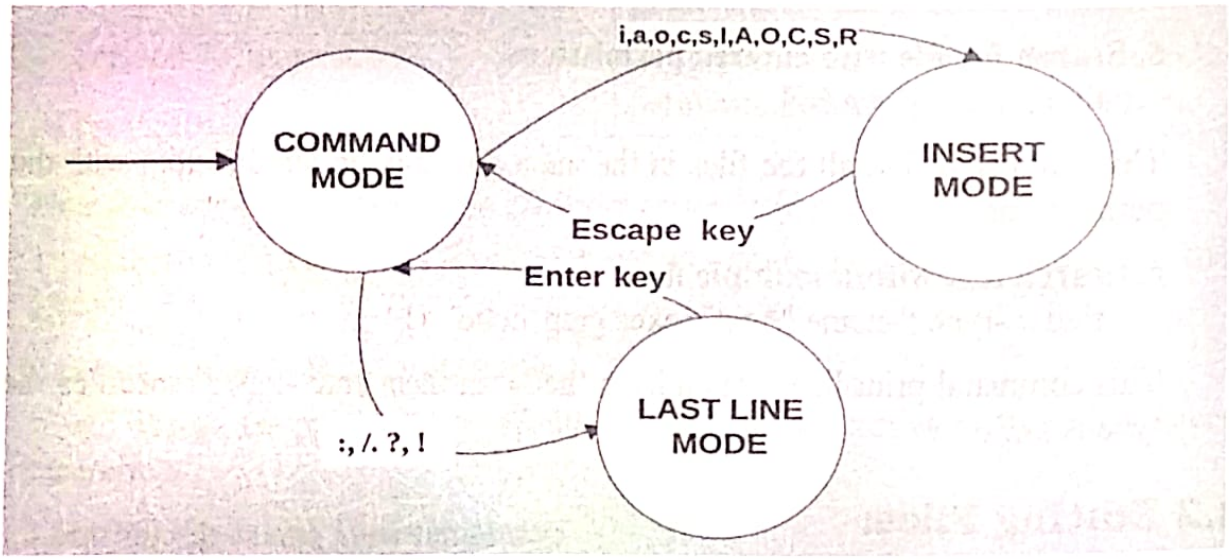
vi is the original editor for Unix. It is very fast, easy to use, and available on virtually every UNIX / LINUX system. The vi commands are the same as several other common LINUX tools. The improved version of vi called vim (vi IMproved) was written and released in 1991. vim is upward-compatible from vi, so that everything in vi works in vim. vim offers more features than vi, but it is also much harder to learn than vi.

vi has three modes, or states that is Command Mode, Insert Mode, Last-line Mode. At any given time it is in exactly one of these modes. With each key that you press, it either stays in its current mode or changes to a different one.

vi is a CASE-SENSITIVE: everything you type must be in the correct case. vi makes a copy of the file in a temporary location to use as its work buffer. Sometimes this copy is readable by other people who know how to find it.

⇨ **How to edit using vi Editor:**

vi editor works as per the image below. The arrows from one state to another are labelled by the keys that cause its state to change. As per the image, when user wish to write or insert (Command mode to Insert mode) then required to press any options as given in the image(i, a, o, c, s, I, A, O, C, S, R) and same way for last line mode(:, /, ?, !).



emacs is a very powerful editor, but it takes more effort to learn how to use it.

pico is the easiest editor to learn and the least powerful.

2.3 Basics of Shell Scripting Programming:

User can use any of the editor like pico, emacs and vi to write the script. There are two types of comments.

1. Single line comment: '#' symbol is used to add single line comment in unix/linux.
2. Multi line comment: ':' and " ' " symbols are used to add multiline comment.

Scanned with CamScanner

Example:

```
# First Shell Script
:'
   echo is used for display the message and goto the next line
   echo -n is used for display the message and -n means not go to the next line.
   and
   read is use to accept the value
'
echo "Hello How are you" # Display Message
echo -n "Enter the Rollno:"  # Display Message
read Rollno              # Accept Rollno from user
echo -n "Enter your Surname:"
read Surname
echo -n "Enter your Name:"
read Name
echo "Full Name is: $Name $Surname"
```

Output:

```
Hello How are you
Enter the Rollno:1
Enter your Surname:Shah
Enter your Name:Harshi
Full Name is: Harshi Shah
```

- **Expr command:**

The expr command in Linux/Unix evaluates a given expression and displays its corresponding output. It is used for the basic operations like addition, subtraction, multiplication, division and modulus on integers. It is also used for the evaluating regular expressions, string operations like substring, length of strings etc.

Syntax:

$expr expression

Example of expr using Console:

Expression	Output	Description
$ expr 12 + 8	20	Addition of two numbers
$ expr 12 - 8	12	Subtraction of two numbers

Scanned with CamScanner

$ expr 12 * 8	96	Multiplication of two numbers
$ expr 12 / 8	1	Division of two numbers
$ expr 12 % 8	4	Modulus of two numbers

⇨ **Example of expr using Shell Script:**

```
# Write a shell script to accept two numbers and display sum.
echo -n "Enter the NO1:"
read no1
echo -n "Enter the NO2:"
read no2
 sum=`expr $no1 + $no2`
 echo "Sum is: " $sum
```

Output:

Enter the NO1:10

Enter the NO2:20

Sum is:30

⇨ **Example of expr to compare values using shell script:**

```
# Write a shell script to accept two numbers and compare it.
echo -n "Enter the NO1:"
read no1
echo -n "Enter the NO2:"
read no2
res1=`expr $no1 = $no2`
echo "Result using = operator is:" $res1
res2=`expr $no1 \> $no2`
echo "Result using > operator:" $res2
res3=`expr $no1 \!= $no2`
echo "Result using != operator:" $res3
```

Output:

Enter the NO1:10

Enter the NO2:20

Result is: 0 (Display 0 if both the numbers are different, otherwise it displays 1).

Result using > operator: 0 (Check 10>20, NO=0 and YES=1)

Scanned with CamScanner

Result using != operator: 1 (Check 10!=20, NO=0 and YES=1)

⇨ **Example of expr for String Operations:**

```
# Write a shell script to accept your name and display length of name.
echo -n "Enter your Name:"
read name
len=`expr length $name`
echo "length is:" $len
sub=`expr substr $name 1 3`
echo $sub
```

Output:

```
Enter your Name:Keyur
length is: 5
Substring is: Key
```

⇨ **Conditional (Decision) Statements:**

There are total 5 conditional statements which can be used in unix / linux programming.

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

1. **if statement:**

This block will process if specified condition is true.

Syntax:

```
if [ expression ]
then
        statement
fi
```

Example:

```
# Write a Shell Script to check two numbers are equal or not.
a=10
b=20
```


Scanned with CamScanner

```
#Check whether they are equal
if [ $a == $b ]
then
  echo "a is equal to b"
fi
#Check whether they are not equal
if [ $a != $b ]
then
  echo "a is not equal to b"
fi
```

Output: a is not equal to b

2. **if-else statement:**

If specified condition is not true in if part then else part will be execute.

Syntax:

```
if [ expression ]
then
        statement1
else
        statement2
fi
```

Example:

```
# Write a Shell Script to check two numbers are equal or not.
a=20
b=20
if [ $a == $b ]
then
    echo "a is equal to b"  #If they are equal then print this.
else
  echo "a is not equal to b"    #else print this
fi
```

Output: a is equal to b

Scanned with CamScanner

3. if..elif..else..fi statement (Else If ladder):

To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.

Syntax

```
if [ expression1 ]
then
        statement1
        statement2
elif [ expression2 ]
then
        statement3
        statement4
else
        statement5
fi
```

Example:

```
: ' Shell Script to accept rollno, name and marks of english, maths and science from user
and display sum, percentage and grade as follows:
# percentage >=80, "A", >=70, "B", >=60, "C" otherwise "F"
echo -n "Enter the Rollno:"  # Display Message
read rollno                  # Accept Rollno from user
echo -n "Enter your Name:"
read name
echo -n "Enter the marks of English :"
read eng
echo -n "Enter the marks of Maths :"
read maths
echo -n "Enter the marks of Science :"
read sci
sum=`expr $eng + $maths + $sci`
echo "Sum is:" $sum
```


Scanned with CamScanner

```
per=`expr $sum / 3`
echo "Per is:"$per
if [ $per -gt 80 ]
then
   echo "Grade A"

elif [ $per -gt 70 ]
then
   echo "Grade B"
else
   echo "Grade C"

fi
```

Output:

```
Enter the Rollno:1
Enter your Name:HARSHI
Enter the marks of English :65
Enter the marks of Maths :97
Enter the marks of Science :88
Sum is: 250
Per is:83
Grade A
```

4. if..then..else..if..then..fi..fi..(Nested if):

Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.

Syntax:

```
if [ expression1 ]
then
  statement1
  statement2
  else
```


Scanned with CamScanner

```
  if [ expression2 ]
  then
    statement3
   fi
fi
```

Example:

```
count=99
if [ $count -eq 100 ] # eq means equal to
then
 echo "Count is 100"
else
 if [ $count -gt 100 ] # gt means greater than
 then
  echo "Count is greater than 100"
 else
 echo "Count is less than 100"
 fi
fi
```

Output: Count is less than 100

5. **switch statement:**

Case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern. When a match is found all of the associated statements until the double semicolon (;;) is executed.

A case will be terminated when the last command is executed. If there is no match, the exit status of the case is zero.

Syntax:

```
case in
  Pattern 1) Statement 1;;
  Pattern n) Statement n;;
esac
```

Example:

Write a shell script to accept two numbers and operator from user and display appropriate output.

Scanned with CamScanner

```
echo -n "Enter the Number 1 :"
read no1
echo -n "Enter the Number 2 :"
read no2
echo -n "Enter the Operator (+,-,*,/) :"
read op
case $op in
  "+") ans=`expr $no1 + $no2`
     echo "Addition of two number is :" $ans ;;
  "-") ans=`expr $no1 - $no2`
     echo "Subtraction of two number is :" $ans ;;
  "*") ans=`expr $no1 \* $no2`
     echo "Multiplication of two number is :" $ans ;;
  "/") ans=`expr $no1 / $no2`
     echo "Division of two number is :" $ans ;;
esac
```

Output:

```
Enter the Number 1 :20
Enter the Number 2 :10
Enter the Operator (+,-,*,/) :/
Division of two number is : 2
```

- **Test Command:**

The test command is used to check file types and compare values. Test is used in conditional execution. It is used for:

- File attributes comparisons
- Perform string comparisons.
- Basic arithmetic comparisons.

Syntax:

```
test [EXPRESSION]
        or
[EXPRESSION]
```


Scanned with CamScanner

Example:

Expression	Output	Description
test 100 -gt 99 && echo "Yes, that's true." || echo "No, that's false."	Yes, that's true.	Conditions is true so it gives Yes, that's true.
test 100 -lt 99 && echo "Yes." || echo "No."	No.	Condition is false so it gives No.
[100 -gt 101] ; echo $?	1	If condition is true than 0 otherwise it gives 1. Here, condition is false.
[100 -gt 99] ; echo $?	0	Condition is true so it gives 0 as output.
["Gujarat" = "Gujarat"]; echo $?	0	Condition is true so it gives 0 as output.
["Gujarat" = "gujarat"]; echo $?	1	Condition is False so it gives 1 as output.
test -f /etc/resolv.conf && echo "File /etc/resolv.conf found." || echo "File /etc/resolv.conf not found."	File /etc/resolv.conf found.	Check the file is available or not and give the output accordingly.

⇨ **Looping Statements, Break and Continue:** There are total 3 looping statements which can be used in unix / linux programming.

→ while statement

→ for statement

→ until statement

While statement: In while loop / statement, it works as per the evalution of condition / command and executes the statement accordingly.

Syntax

```
while command
do
  Statement to be executed
Done
```

Example:

```
i=1
while [ $i -le 10 ]
```


Scanned with CamScanner

```
do
  echo $i
  i=`expr $i + 1`
done
```

Output: 12345678910

For statement: For loop is operate as per the start and end values. It will execute the statement while variable value not reach it to the end value.

Syntax

```
for (( expr1; expr2; expr3 ))
do
command1
command2
..
Done
```

Example:

```
for (( i=1; i<=10; i++ ))
do
  echo -n $i
done
```

Output: 12345678910

⇨ **Until statement:**

The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

Syntax:

```
until command
do
  Statement to be executed until command is true
done
```

Example:

```
a=1
until [ ! $a -lt 10 ]
do
  echo -n $a
```


Scanned with CamScanner

```
    a=`expr $a + 1`
  done
```

Output: 12345678910

- **Break and Continue:**

break: The break statement is used to terminate the loop and can be used within a while, for, until, and select loops. It is used to alter the flow of loop statements.

Syntax:

```
  break
```

Example:

```
for (( i=1; i<=10; i++ ))
do
    if [ $i -eq 5 ]
    then
        break
    fi
    echo -n $i
done
```

Output: 1234

- **continue:** continue is a command which is used to skip the current iteration in for, while and until loop.

Syntax:

```
  continue
```

Example:

```
for (( i=1; i<=10; i++ ))
do
    if [ $i -eq 5 ]
    then
        continue
    fi
    echo -n $i
done
```

Output: 1234678910

☺ ☺ ☺ ☺ ☺

Scanned with CamScanner

UNIT-3

Simple Filters, Filters using regular expression, Advanced Filters

- **Simple Filters:** Paginating files –pr, Splitting a file vertically–cut, Pasting files-paste, Ordering a file-sort, Locate repeated and non repeatedlines-uniq, Translating characters–tr
- Filters using regular expression: Searching for pattern-grep, stream editior –sed
- Advanced filters: Simple awkfiltering, Comparison operators, Variables, Built invariables, Control flow, Looping

Scanned with CamScanner

Unit – 3 Filters

3.1 Simple Filters:

⇨ Paginating files –pr:

In Linux/Unix pr command is used to prepare a file for printing by adding suitable footers, headers, and the formatted text. **pr** command actually adds 5 lines of margin both at the top and bottom of the page. The header part shows the date and time of the last modification of the file with the file name and the page number.

Syntax:

pr [options][filename]

⇨ Splitting a file vertically–cut, Pasting files-paste:

1) **Split :**

Split command in Linux is used to **split large files into smaller files**. It splits the files into 1000 lines per file(by default) and even allows users to change the number of lines as per requirement.

The names of the files are PREFIXaa, PREFIXab, PREFIXac, and so on. By default the PREFIX of files name is x and the default size of each split file is 1000 lines per file and both the parameters can be changed with ease. It is generally used with log and archive files as they are very large and have a lot of lines, So in order to break them into small files for analysis split command is used.

Syntax:

split [options] name_of_file prefix_for_new_files

2) **Cut:**

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error. If more than one file name is provided then data from each file is **not precedes** by its file name.

Syntax:

cut OPTION... [FILE]...

Let us consider two files having name **state.txt** and **capital.txt** contains 5 names of the Indian states and capitals respectively.

$ cat state.txt

Scanned with CamScanner

Gujarat

Rajasthan

Assam

Bihar

Chhattisgarh

Without any option specified it displays error.

$ cut state.txt

cut: you must specify a list of bytes, characters, or fields
Try 'cut --help' for more information.

Options and their Description with examples:

1. -b(byte): To extract the specific bytes, you need to follow -b option with the list of byte numbers separated by comma. Range of bytes can also be specified using the hyphen(-). It is necessary to specify list of byte numbers otherwise it gives error. **Tabs and backspaces** are treated like as a character of 1 byte.

List without ranges
$ cut -b 1,2,3 state.txt
Guj
Raj
Ass
Bih
Chh

2. -c (column): To cut by character use the -c option. This selects the characters given to the -c option. This can be a list of numbers separated comma or a range of numbers separated by hyphen(-). **Tabs and backspaces** are treated as a character. It is necessary to specify list of character numbers otherwise it gives error with this option.
Syntax:
$cut -c [(k)-(n)/(k),(n)/(n)] filename

3) **Paste:**
Paste command is one of the useful commands in Unix or Linux operating system. It is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by **tab** as delimiter, to the standard output. When no file is specified, or put dash ("-") instead of file name, paste reads from standard input and gives output as it is until a interrupt command **[Ctrl-c]** is given.

Scanned with CamScanner

Syntax:
paste [OPTION]... [FILES]...

Let us consider three files having name **state, capital** and **number.**

state and *capital* file contains 5 names of the Indian states and capitals respectively. *number* file contains 5 numbers.

$ cat state
Gujarat
Assam
Andhra Pradesh
Bihar
Chhattisgrah

$ cat capital
Gandhinagar
Dispur
Hyderabad
Patna
Raipur

Without any option paste merges the files in parallel. The paste command writes corresponding lines from the files with tab as a deliminator on the terminal.

$ paste number state capital

1	Gujarat	Gandhinagar
2	Assam	Dispur
3	Andhra Pradesh	Hyderabad
4	Bihar	Patna
5	Chhattisgrah	Raipur

n the above command three files are merges by paste command.

⇨ **Options:**

1. -d (delimiter): Paste command uses the tab delimiter by default for merging the files. The delimiter can be changed to any other character by using the **-d** option. If more than one character is specified as delimiter then paste uses it in a circular fashion for each file line separation.

Only one character is specified
$ paste -d "|" number state capital
1| Gujarat | Gandhinagar

Scanned with CamScanner

2|Assam|Dispur
3|Andhra Pradesh|Hyderabad

4|Bihar|Patna

5|Chhattisgrah|Raipur

More than one character is specified:

$ paste -d "|," number state capital
1| Gujarat, Gandhinagar
2|Assam,Dispur
3|Andhra Pradesh,Hyderabad
4|Bihar,Patna
5|Chhattisgrah,Raipur

First and second file is separated by '|' and second and third is separated by ','.

After that list is exhausted and reused.

2. -s (serial): We can merge the files in sequentially manner using the -s option. It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab. And these single lines are separated by newline.

$ paste -s number state capital

1	2	3	4	5
Gujarat	Assam	Andhra Pradesh	Bihar	Chhattisgrah
Gandhinagar	Dispur	Hyderabad	Patna	Raipur

In the above command, first it reads data from **number** file and merge them into single line with each line separated by tab. After that newline character is introduced and reading from next file i.e. **state** starts and process repeats again till all files are read.

❖ Filters in Linux:

Filters are programs that take plain text(either stored in a file or produced by another program) as standard input, transforms it into a meaningful format, and then returns it as standard output. Linux has a number of filters. Some of the most commonly used filters are explained below:

1. SORT:

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in the sort command can also be used to sort numerically.

SORT command sorts the contents of a text file, line by line.

Scanned with CamScanner

- sort is a standard command-line program that prints the lines of its input or concatenation of all files listed in its argument list in sorted order.
- The sort command is a command-line utility for sorting lines of text files. It supports sorting alphabetically, in reverse order, by number, by month, and can also remove duplicates.
- The sort command can also sort by items not at the beginning of the line, ignore case sensitivity, and return whether a file is sorted or not. Sorting is done based on one or more sort keys extracted from each line of input.
- By default, the entire input is taken as the sort key. Blank space is the default field separator.

Sorting a file: Now use the sort command

Syntax :
$ sort filename.txt

2. **uniq :**

The **uniq** command in Linux is a command-line utility that reports or filters out the repeated lines in a file.

In simple words, **uniq** is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines. **uniq** filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file.

Syntax of uniq Command :

/...syntax of uniq...//

$uniq [OPTION] [INPUT[OUTPUT]]

The syntax of this is quite easy to understand. Here, **INPUT** refers to the input file in which repeated lines need to be filtered out and if INPUT isn't specified then **uniq** reads from the standard input. **OUTPUT** refers to the output file in which you can store the filtered output generated by **uniq** command and as in the case of INPUT if OUTPUT isn't specified then **uniq** writes to the standard output.

Now, let's understand the use of this with the help of an example. Suppose you have a text file named kt.txt which contains repeated lines that needs to be omitted. This can simply be done with uniq.

3. **Translating characters–tr**

tr is short for "translate". It is a member of the GNU coreutils package. Therefore, it's available in all Linux distros.

Scanned with CamScanner

The *tr* command reads a byte stream from standard input (*stdin*), translates or deletes characters, then writes the result to the standard output (*stdout*).

The usage syntax of *tr* is pretty straightforward:

Syntax:

command | tr <'old'> <'new'>

Since *tr* doesn't support reading a file directly, if we want to apply it to a text file, we need to pipe the file content to *tr* or redirect the file to *stdin*.

We can use *tr* to perform text transformations like:

- character case conversion
- squeezing repeating characters
- deleting specific characters
- basic text replacement

⇨ **Change Case:**

The 'tr' command can change case.

Syntax:

command | tr <'old'> <'new'>

3.2 Filters using regular expression: Searching for pattern-grep:

1) Grep:

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).

Syntax:

grep [options] pattern [files]

Options Description:

-c : This prints only a count of the lines that match a pattern
-h : Display the matched lines, but do not display the filenames.
-i : Ignores, case for matching
-l : Displays list of a filenames only.
-n : Display the matched lines and their line numbers.
-v : This prints out all the lines that do not matches the pattern
-e exp : Specifies expression with this option. Can use multiple times.
-f file : Takes patterns from file, one per line.

Scanned with CamScanner

-E : Treats pattern as an extended regular expression (ERE)
-w : Match whole word
-o : Print only the matched parts of a matching line, with each such part on a separate output line.
-A n : Prints searched line and nlines after the result.
-B n : Prints searched line and n line before the result.
-C n : Prints searched line and n lines after before the result.

The grep command stands for "global regular expression print", and it is one of the most powerful and commonly used commands in Linux.

grep searches one or more input files for lines that match a given pattern and writes each matching line to standard output. If no files are specified, grep reads from the standard input, which is usually the output of another command.

In this article, we will show you how to use the grep command through practical examples and detailed explanations of the most common GNU grep options.

2) stream editior –sed:

sed operates on a stream of text that it reads from either a text file or from standard input (STDIN). This means that you can send the output of another command directly into sed for editing, or you can work on a file that you've already created.

You should also be aware that sed outputs everything to standard out (STDOUT) by default. That means that, unless redirected, sed will print its output to the screen instead of saving it in a file.

The basic usage is:

sed [options] commands [file-to-edit]

In this tutorial, you'll use a copy of the BSD Software License to experiment with sed. On Ubuntu, execute the following commands to copy the BSD license file to your home directory so you can work with it:

- cd
- cp /usr/share/common-licenses/BSD.

3.3 Advanced filters:

➢ **Awk Command:**

In awk, we are having the functionality to read the file. But here, we are using integer variables to read the file.

Below are the integer information and meaning if they are using in awk commands.

- **$0:** read the complete file or input text.

Scanned with CamScanner

- **$1:** read the first field.
- **$2:** read the second field.
- **$n:** read the nth field.

Note: If we are not using any separator keyword "-F" in the awk command. Then the awk command considers the space in the text file or input data and separates the result as per the $variable provided in the same awk command.

Syntax:

```
cat file.txt | awk '{print $1}'
```

Explanation:

- In file.txt, we are having the sample data.
- If we will use $0 then entire data will read.
- If we will use $1 then only first column data will filter out.

Comparison operators:

There are various operators supported by each shell. We will discuss in detail about Bourne shell (default shell) in this chapter.

We will now discuss the following operators –

- Arithmetic Operators
- Relational Operators
- Boolean Operators
- String Operators
- File Test Operators

Bourne shell didn't originally have any mechanism to perform simple arithmetic operations but it uses external programs, either **awk** or **expr**.

The following example shows how to add two numbers –

```
#!/bin/sh
val=`expr 2 + 2`
echo "Total value : $val"
```

The above script will generate the following result –

```
Total value : 4
```

The following points need to be considered while adding –

- There must be spaces between operators and expressions. For example, 2+2 is not correct; it should be written as 2 + 2.

Scanned with CamScanner

- The complete expression should be enclosed between ‘ ‘, called the backtick.

Variables:

Variables are used to store any value temporary in any programming language. Defining the variable in awk command is similar to bash scripting language and it works like bash when the shell variable is used with a single quote and double quote. Awk command has many built-in variables for various purposes. How user-defined, built-in and shell variables can be used in awk command is shown in this tutorial by using different examples.

Example -1: Defining and printing variable:

`awk` command uses ‘-v’ option to define the variable. In this example, the **myvar** variable is defined in the `awk` command to store the value, **“AWK variable”** that is printed later. Run the following command from the terminal to check the output.

```
$ echo | awk -v myvar='AWK variable' '{print myvar}'
```

Example – 2: Using shell variable in awk with a single quote and double quote:

The example shows how the shell variable can be used `awk` command. Here, a shell variable, **myvar** is declared with the value, **“Linux exp”** in the first command. ‘$’ symbol is used with a shell variable name to read the value. The second command reads the variable, **$myval** with a single quote(‘) and the third command reads the variable $myvar with double quote(“) in the `awk` statement.

```
$ myvar="Linux exp"
$ echo | awk -v awkvar='$myvar' '{ print awkvar; }'
$ echo | awk -v awkvar="$myvar" '{ print awkvar; }'
```

It is shown in the output that the value of $myvar can’t be read when it is enclosed with a single quote (‘) and the output is **$myvar**. The value of **$myvar** is printed when it is enclosed with a double quote (“).

Example – 3: Reading ARGC variable in awk

ARGC variable is used to count the total number of command line arguments. Three command line arguments variables (t1, t2, t3) are passed in the following awk script. Here, the total number of arguments with the script is 4. Run the script from the terminal.

```
$ awk 'BEGIN{print "Total arguments=",ARGC}' t1 t2 t3
```


Scanned with CamScanner

➢ Using built-in variable:

Using built-in variable, FS and field separator option with awk command.
FS variable is used in awk command as a field separator. Space is used as a default value of FS. The following command will read the file **customer.txt** using space as field separator and print the file content. Run the command from the terminal.

$ awk FS customer.txt

Using built-in variable, NR with awk command:

NR variable is used in awk command to count the total number of records or lines of a file. Create a text file named **student.txt** to test the function of this variable.

student.txt

Name	Batch	Semester
Ravi	20	3
Tina	22	1
Gopi	1	8
Sonu	15	8

The following awk script will print the first three lines of **product.txt** file. Here, a condition is added by using the NR variable. The command will print those lines where the NR value is less than 4. Run the script from the terminal.

Output:

The following output will appear after running the script.

$ cat student.txt | awk 'NR < 4'

➢ Awk- Control flow:

Like other programming languages, AWK provides conditional statements to control the flow of a program. This chapter explains AWK's control statements with suitable **Examples.**

1. If statement:

It simply tests the condition and performs certain actions depending upon the condition. Given below is the syntax of **if** statement –

Syntax:

if (condition)

action

We can also use a pair of curly braces as given below to execute multiple actions –

Syntax

if (condition) {

Scanned with CamScanner

action-1

action-1

.

.

action-n

}

2. **If Else Statement:**

In **if-else** syntax, we can provide a list of actions to be performed when a condition becomes false.

The syntax of **if-else** statement is as follows –

Syntax

if (condition)

action-1

else

action-2

3. **If-Else-If Ladder:**

We can easily create an **if-else-if** ladder by using multiple **if-else** statements.

➢ **Awk – Looping:**

Awk scripts have three main sections: the optional BEGIN and END functions and the functions you write that are executed on each record. In a way, the main body of an awk script is a loop, because the commands in the functions run for each record. However, sometimes you want to run commands on a record more than once, and for that to happen, you must write a loop.

There are several kinds of loops, each serving a unique purpose.

1. **While loop:**

A **while** loop tests a condition and performs commands *while* the test returns *true*. Once a test returns *false*, the loop is broken.

2. **Do while loop:**

The **do while** loop performs commands after the keyword **do**. It performs a test afterward to determine whether the stop condition has been met. The commands are repeated only *while* the test returns true (that is, the end condition has *not* been met). If a test fails, the loop is broken because the end condition has been met.

3. **For loops:**

There are two kinds of **for** loops in awk.

Scanned with CamScanner

One kind of **for** loop initializes a variable, performs a test, and increments the variable together, performing commands while the test is true.

➢ **Loops:**

Loops are a vital part of any programming language, and awk is no exception. Using loops can help you control how your awk script runs, what information it's able to gather, and how it processes your data. Our next article will cover switch statements, **continue**, and **next**.

☺ ☺ ☺ ☺ ☺

Scanned with CamScanner

UNIT-4

Compressing, decompressing and achieving files

- Compressing , decompressing and achieving files: Gzip, gunzip, Tar, Zip and unzip
- Environment variables: Environment variables, Alias, Inline command editing, miscellaneous features, Initialization script.
- Communication commands: Finger, Talk, Mesg, Mailx, Pine, Write, Wall

Scanned with CamScanner

In the above example command this would compress the file.txt file as file.txt.gz in the current directory.

⇨ ***Uncompressing using Gunzip:***

gunzip file.txt.gz

In the above example command it would extract the file.txt from file.txt.gz.

⇨ **Zip and unzip:**

Zip and Unzip are compression and decompression utilities respectively.

Zip puts one or more compressed files into a single "zip file" along with information about the files, including the name, path if requested, date and time last modified, protection, and check information to verify the fidelity of each entry.

zip newfile oldfile

This commands takes an existing file, here oldfile, and zips it under the name specified in newfile. Remember that the default suffix, if none is specified, will be .zip. This allows specifying suffixes other than ".zip".

To zip up all of the files in your current directory, type:

zip mydir *

This will create the file "mydir.zip" and put all the files in the current directory in mydir.zip in a compressed form. This will not include any subdirectories in the current directory.

To zip up an entire directory, including subdirectories, the command:

zip -r mydir

will create the file "mydir.zip" containing all the files and directories in the directory "mydir" that is in the current directory. The "r" option means recursive through the directory structure.

Unzip is the command to decompress these files.

To unzip a file named myfile.zip

unzip myfile.zip

This will unzip all the files in the current directory that were zipped into the file.

4.2 Environment Variables:

➢ System Variables:

These variables as system-level variables, as they are used to configure your entire system, setting values such as the location of executable commands on your system

Some of the system variables are as follows:

Scanned with CamScanner

Unit -4 Compressing, decompressing and achieving files

4.1 Compressing and Uncompressing files:

It is useful to compress files into one file so that they use less disk space and download faster via the Internet.

There is distinction between an archive file and a compressed file. An archive file is a collection of files and directories that are stored in one file. The archive file is not compressed and it uses the same amount of disk space as all the individual files and directories combined.

A compressed file is a collection of files and directories that are stored in one file and stored in a way that uses less disk space than all the individual files and directories combined. You can compress files that you do not use very often or files that you want to save but do not use anymore. You can even create an archive file and then compress it to save disk space.

An archive file is not compressed, but a compressed file can be an archive file.

- **Command: TAR**

The compressed file will have the extension of .tar

A ".tar" file is not a compressed files, it is actually a collection of files within a single file uncompressed.

Creating a tar file:

tar -cvwf file1.tar myfile.txt

The above command would create a file named file1.tar in the directory you currently are in.

Extracting the files from a tar file:

tar -xvwf file1.tar

In the above example command the system would uncompress (untar) the file1.tar file in the current directory.

- **Gzip and Gunzip:**

It is used to Compress or uncompress files.

⇨ ***Compressing using Gzip:***

gzip file.txt

Scanned with CamScanner

In the above example command this would compress the file.txt file as file.txt.gz in the current directory.

⇨ ***Uncompressing using Gunzip:***

gunzip file.txt.gz

In the above example command it would extract the file.txt from file.txt.gz.

⇨ **Zip and unzip:**

Zip and Unzip are compression and decompression utilities respectively.

Zip puts one or more compressed files into a single "zip file" along with information about the files, including the name, path if requested, date and time last modified, protection, and check information to verify the fidelity of each entry.

zip newfile oldfile

This commands takes an existing file, here oldfile, and zips it under the name specified in newfile. Remember that the default suffix, if none is specified, will be .zip. This allows specifying suffixes other than ".zip".

To zip up all of the files in your current directory, type:

zip mydir *

This will create the file "mydir.zip" and put all the files in the current directory in mydir.zip in a compressed form. This will not include any subdirectories in the current directory.

To zip up an entire directory, including subdirectories, the command:

zip -r mydir

will create the file "mydir.zip" containing all the files and directories in the directory "mydir" that is in the current directory. The "r" option means recursive through the directory structure.

Unzip is the command to decompress these files.

To unzip a file named myfile.zip

unzip myfile.zip

This will unzip all the files in the current directory that were zipped into the file.

4.2 Environment Variables:

➢ **System Variables:**

These variables as system-level variables, as they are used to configure your entire system, setting values such as the location of executable commands on your system

Some of the system variables are as follows:

Scanned with CamScanner

PS1 and PS2:

The **PS1** and **PS2** variables contain the primary and secondary prompt symbols, respectively. The primary prompt symbol for the BASH shell is a dollar sign ($).

You can change the prompt symbol by assigning a new set of characters to the **PS1** variable. In the next example, the shell prompt is changed to the -> symbol:

```
$ PS1= '->'
-> export PS1
->
```

The **PS2** variable holds the secondary prompt symbol, which is used for commands. The default secondary prompt is >. The added command lines begin with the secondary prompt instead of the primary prompt. You can change the secondary prompt just as easily as the primary prompt, as shown here:

```
$ PS2="@"
```

⇨ **PATH:**

The **PATH** variable contains a series of directory paths separated by colons. Each time a command is executed, the paths listed in the **PATH** variable are searched one by one for that command.

For example, the **ls** command resides on the system in the directory **/bin**. This directory path is one of the directories listed in the **PATH** variable. Each time you execute the ls command, this path is searched and the ls command located.

The system defines and assigns **PATH** an initial set of pathnames. In Linux, the initial pathnames are **/bin** and **/usr/bin**.

The shell can execute any executable file, including programs and scripts you have created. For this reason, the **PATH** variable can also reference your working directory; so if you want to execute one of your own scripts or programs in your working directory, the shell can locate it.

```
$ echo $PATH
/bin:/usr/sbin:
```

You can add any new directory path you want to the **PATH** variable. You can place these new shell script commands in a directory you create and then add that directory to the **PATH** list.

For example:

The user user1 adds a new directory, called mydir, to the **PATH**. In this example, an evaluation of **HOME** is also used to add the user's **home** directory in the new directory's pathname.

Scanned with CamScanner

$ PATH=$PATH:$HOME/mydir:

$ export PATH

$ echo $PATH

/bin:/usr/bin::/home/user1/mydir

⇨ **HOME:**

The **HOME** variable contains the pathname of your home directory. Your home directory is determined by the parameter administrator when your account is created.

In the next example, the **echo** command displays the contents of the **HOME** variable:

$ echo $HOME

/home/user1

The **HOME** variable is often used when you need to specify the absolute pathname of your home directory.

⇨ **LOGNAME:**

This variable is assigned the name of the user.

Both LOGNAME and USER should be set to the username.

Examples:

LOGNAME=user1

To display the username

echo $LOGNAME

➢ **Command line arguments:**

Command line argument is used to get the input from the user, when user runs the shell script. Command line arguments are treated as special variables within the script, the reason I am calling them variables is because they can be changed with the shift command.

The command line arguments are enumerated in the following manner $0, $1, $2, $3, $4, $5, $6, $7, $8 and $9. $0 is special in that it corresponds to the name of the script itself. $1 is the first argument; $2 is the second argument and so on.

gedit myfile.sh

it will open a text editor

add the following code

```
echo $1
echo $2
```

save the file and run your shell script as sh myfile.sh first second

Scanned with CamScanner

output will be

first

second

As well as the command line arguments there are some special builtin variables:

1) **$#**

$# counts the number of command line arguments. It is useful for controlling if conditions and loop constructs that need to process each parameter.

```
if [ $# ]
then
  echo "command line arguments found"
else
  echo "no command line arguments supplied"
fi
```

2) **$@**

It expands to all the parameters separated by spaces. It is useful for passing all the parameters to some other function or program.

```
echo "Enter the word"
read w
for i in $@
do
grep $w $i
done
```

run this script as sh filename.sh file1 file2

file1 and file2 should be created before running this code.

This script will search for a word in multiple files entered through command line arguments.

3) **$$** expands to the process id of the shell innovated to run the script. It is useful for creating unique temporary filenames relative to this instantiation of the script.

4.3 Communication Commands:

⇨ **finger:**

It displays information about the user. It displays more extended information than who.

$ finger [username]

finger -b -p user1 - Would display the following information about the user user1.

Scanned with CamScanner

Login name: user1 In real life: User1

On since Jan 01 12:30:16 on pts/1 from domain.user1.com

⇨ **talk:**

This command is used communicate to another user.

talk person [tty]

Talk is a visual program which copies lines from your terminal to that of another user.

⇨ **mesg:**

mesg command is used to give write access to your terminal

mesg [y|n]

mesg manages the access to your terminal by others. It's typically used to allow or disallow other users to write to your terminal

mesg y Allow write access to your terminal.

mesg n Disallow write access to your terminal.

⇨ **mailx:**

It is used for sending and receiving mail through interactive message processing system.

mailx info@computerworld.com - Start a new mail message to be sent to support at Computer World.

⇨ **pine:**

Pine is a command line program for Internet News and Email.

Syntax:

pine [address , address]

address Send mail to address this will open pine directly into the message composer.

⇨ **Write:**

write command in Linux is used to send a message to another user. The write utility allows a user to communicate with other users, by copying lines from one user's terminal to others. When you run the write command, the user you are writing to gets a message of the form:

Message from yourname@yourhost on yourtty at hh:mm ...

Any further lines the user enter will be copied to the specified user's terminal. If the other user wants to reply, they must run write as well. When you are done, type an end-of-file or interrupt character. The other user will see the message 'EOF' indicating that the conversation is over.

Scanned with CamScanner

Syntax:

write user [tty]

write command without any option: It will print the general syntax of the write. With the help of this, the user will get a generalized idea about how to use this command since there nothing like help option for the write command.

⇨ **Wall:**

wall is a command-line utility that displays a message on the terminals of all logged-in users. The messages can be either typed on the terminal or the contents of a file. wall stands for write all, to send a message only to a specific user use the write command.

Usually, system administrators send messages to announce maintenance and ask users to log out and close all open programs. The messages are shown to all logged-in users with a terminal open. Users using a graphical desktop environment with no terminal open will not see the messages. Each user can control the write access to its terminal with the mesg utility. When the superuser invokes the wall command, all users receive the messages, no matter their mesg settings.

Broadcasting a Message:

The syntax for the wall command is as follows:

wall [OPTIONS] [<FILE>|<MESSAGE>]

If no file is specified wall reads the message from the standard input.

The most straightforward way to broadcast a message is to invoke the wall command with the message as the argument.

Scanned with CamScanner

Practical

1. **Check the output of the following commands: date, ls, who, cal, ps, wc, cat, uname, pwd, mkdir, rmdir, cd, cp, rm, mv, diff, chmod, grep, sed, head, tail,cut, paste, sort, find, man.**

1. **Ls:** This command will list the files stored in a directory. To see a brief, multi-column list of the files in the current directory, enter: ls

   To also see "dot" files (configuration files that begin with a period, such as .login), enter: ls -a

   To see the file permissions, owners, and sizes of all files, enter: ls -la

   If the listing is long and scrolls off your screen before you can read it, combine ls with the less utility, for example: ls -la | less

2. **Who:** The w and who commands are similar programs that list all users logged into the computer. If you use w, you also get a list of what they are doing. If you use who, you also get the IP numbers or computer names of the terminals they are using.

3. **Cal:** This command will print a calendar for a specified month and/or year.

   To show this month's calendar, enter: cal

   To show a twelve-month calendar for 2021, enter: cal 2021

   To show a calendar for just the month of June 1990, enter: cal 6 1990

4. **Ps:** The ps command displays information about programs (that is, processes) that are currently running. Entered without arguments, it lists basic information about interactive processes you own. However, it also has many options for determining what processes to display, as well as the amount of information about each. Like lp and lpr, the options available differ between BSD and System V implementations. For example, to view detailed information about all running processes, in a BSD system, you would use ps with the following arguments: ps -alxww

   To display similar information in System V, use the arguments: ps -elf

5. **Wc:** wc stands for word count. As the name implies, it is mainly used for counting purpose. It is used to find out number of lines, word count, byte and characters count in the files specified in the file arguments. By default it displays four-columnar output.

   First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.

Scanned with CamScanner

Syntax:

wc [OPTION]... [FILE]...

Example:

wc /proc/cpuinfo

Output:

448 3632 22226 /proc/cpuinfo

Where, 448 is the number of lines.3632 is the number of words. 22226 is the number of characters.

6. **Cat:** This command outputs the contents of a text file. You can use it to read brief files or to concatenate files together. To append file1 onto the end of file2, enter:cat file1 >> file2 To view the contents of a file named myfile, enter: cat myfile

   Because cat displays text without pausing, its output may quickly scroll off your screen. Use the less command (described below) or an editor for reading longer text files.

7. **Uname:** uname is a command-line utility that prints basic information about the operating system name and system hardware.

8. **Pwd:** This command reports the current directory path. Enter the command by itself: pwd

9. **Mkdir:** This command will make a new subdirectory. To create a subdirectory named mystuff in the current directory, enter: mkdir mystuff

   To create a subdirectory named morestuff in the existing directory named /tmp, enter:

   mkdir /tmp/morestuff

10. **Rmdir:** This command will remove a subdirectory. To remove a subdirectory named oldstuff, enter: rmdir oldstuff

11. **Cd:** This command changes your current directory location. By default, your Unix login session begins in your home directory. To switch to a subdirectory (of the current directory) named myfiles, enter: cd myfiles

    To switch to a directory named /home/dvader/empire_docs, enter:

    cd /home/dvader/empire_docs

    To move to the parent directory of the current directory, enter:

    cd ..

    To move to the root directory, enter:

    cd /

    To return to your home directory, enter:

    cd

Scanned with CamScanner

12. **Cp:** This command copies a file, preserving the original and creating an identical copy. If you already have a file with the new name, cp will overwrite and destroy the duplicate. For this reason, it's safest to always add -i after the cp command, to force the system to ask for your approval before it destroys any files. The general syntax for cp is:

    cp -i oldfile newfile

    To copy a file named meeting1 in the directory /home/dvader/notes to your current directory, enter:

    cp -i /home/dvader/notes/meeting1 .

    The . (period) indicates the current directory as destination, and the -i ensures that if there is another file named meeting1 in the current directory, you will not overwrite it by accident.

    To copy a file named oldfile in the current directory to the new name newfile in the mystuff subdirectory of your home directory, enter:

    cp -i oldfile ~/mystuff/newfile

    The ~ character (tilde) is interpreted as the path of your home directory.

13. **Rm:** This command will remove (destroy) a file. You should enter this command with the -i option, so that you'll be asked to confirm each file deletion. To remove a file named junk, enter:

    rm -i junk

    Note: Using rm will remove a file permanently, so be sure you really want to delete a file before you use rm.

14. **Mv:** This command will move a file. You can use mv not only to change the directory location of a file, but also to rename files. Unlike the cp command, mv will not preserve the original file. To rename a file named oldname in the current directory to the new name newname, enter:

    mv -i oldname newname

15. **Diff:** diff stands for difference. This command is used to display the differences in the files by comparing the files line by line.

    Syntax :

    diff [options] File1 File2

16. **Chmod:** This command changes the permission information associated with a file. Every file (including directories, which Unix treats as files) on a Unix system is stored with records indicating who has permission to read, write, or execute the file, abbreviated as r, w, and x. These permissions are broken down for three categories of user: first, the owner of the file; second, a group with which both the user and the file may be associated;

Scanned with CamScanner

and third, all other users. These categories are abbreviated as u for owner (or user), g for group, and o for other. To allow yourself to execute a file that you own named myfile, enter:

chmod u+x myfile

To allow anyone who has access to the directory in which myfile is stored to read or execute myfile, enter:

chmod o+rx myfile

17. Grep: prints lines matching a certain pattern.

grep <pattern> filename

The simplest possible example of grep is simply:

grep "boo" a_file

In this example, grep would loop through every line of the file "a_file" and print out every line that

contains the word 'boo':

boot

book

booze

boots

18. Sed: stream editor. sed performs basic text transformations on an input stream (a file or input from a pipeline) in a single pass through the stream, so it is very efficient. However, it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editor.

19. Head: The head command, as the name implies, print the top N number of data of the given input. By default, it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

Syntax:

head [OPTION]... [FILE]...

Let us consider two files having name state.txt and capital.txt contains all the names of the Indian states and capitals respectively.

$ cat state.txt

Andhra Pradesh

Arunachal Pradesh

Assam

Bihar

Chhattisgarh

Scanned with CamScanner

Goa
Gujarat
Haryana
Himachal Pradesh
Jammu and Kashmir
Jharkhand
Karnataka
Kerala
Madhya Pradesh
Maharashtra
Manipur
Meghalaya
Mizoram
Nagaland
Odisha
Punjab
Rajasthan
Sikkim
Tamil Nadu
Telangana
Tripura
Uttar Pradesh
Uttarakhand
West Bengal

Without any option, it displays only the first 10 lines of the file specified.

Example:

$ head state.txt

Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh
Goa
Gujarat

Scanned with CamScanner

Haryana
Himachal Pradesh
Jammu and Kashmir

-n num: Prints the first 'num' lines instead of first 10 lines. num is mandatory to be specified in command otherwise it displays an error.

$ head -n 5 state.txt
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh

-c num: Prints the first 'num' bytes from the file specified. Newline count as a single character, so if head prints out a newline, it will count it as a byte. num is mandatory to be specified in command otherwise displays an error.

$ head -c 6 state.txt
Andhra

-q: It is used if more than 1 file is given. Because of this command, data from each file is not precedes by its file name.

-v: By using this option, data from the specified file is always preceded by its file name.

20. Tail: The tail command, as the name implies, print the last N number of data of the given input. By default it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is precedes by its file name.

Syntax:
tail [OPTION]... [FILE]...

$ tail state.txt
Odisha
Punjab
Rajasthan
Sikkim
Tamil Nadu
Telangana
Tripura
Uttar Pradesh

Scanned with CamScanner

Uttarakhand

West Bengal

-n num: Prints the last 'num' lines instead of last 10 lines. num is mandatory to be specified in command otherwise it displays an error. This command can also be written as without symbolizing 'n' character but '-' sign is mandatory.

c num: Prints the last 'num' bytes from the file specified. Newline count as a single character, so if tail prints out a newline, it will count it as a byte. In this option it is mandatory to write -c followed by positive or negative num depends upon the requirement. By +num, it display all the data after skipping num bytes from starting of the specified file and by -num, it display the last num bytes from the file specified.

-q: It is used if more than 1 file is given. Because of this command, data from each file is not precedes by its file name.

-f: This option is mainly used by system administration to monitor the growth of the log files written by many Unix program as they are running. This option shows the last ten lines of a file and will update when new lines are added. As new lines are written to the log, the console will update with the new lines. The prompt doesn't return even after work is over so, we have to use the interrupt key to abort this command. In general, the applications writes error messages to log files. You can use the -f option to check for the error messages as and when they appear in the log file.

$ tail -f logfile

-v: By using this option, data from the specified file is always preceded by its file name.

–version: This option is used to display the version of tail which is currently running on your system.

21. **Cut:** The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by byte position, character and field.

Syntax:

cut OPTION... [FILE]...

-b(byte): To extract the specific bytes, you need to follow -b option with the list of byte numbers separated by comma. Range of bytes can also be specified using the hyphen(-).

-c (column): To cut by character use the -c option. This selects the characters given to the -c option. This can be a list of numbers separated comma or a range of numbers separated by hyphen(-).

Scanned with CamScanner

-f (field): -c option is useful for fixed-length lines. Most unix files doesn't have fixed-length lines. To extract the useful information you need to cut by fields rather than columns.

–complement: As the name suggests it complement the output. This option can be used in the combination with other options either with -f or with -c.

–output-delimiter: By default the output delimiter is same as input delimiter that we specify in the cut with -d option. To change the output delimiter use the option –output-delimiter="delimiter".

–version: This option is used to display the version of cut which is currently running on your system.

22. **Paste:** Paste command is one of the useful commands in Unix or Linux operating system. It is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by tab as delimiter, to the standard output. When no file is specified, or put dash ("-") instead of file name, paste reads from standard input and gives output as it is until a interrupt command [Ctrl-c] is given.

Syntax:

paste [OPTION]... [FILES]...

-d (delimiter): Paste command uses the tab delimiter by default for merging the files.

-s (serial): We can merge the files in sequentially manner using the -s option.

–version: This option is used to display the version of paste which is currently running on your system.

23. **Sort:** ort command is helpful to sort/order lines in text files. You can sort the data in text file and display the output on the screen, or redirect it to a file. Based on your requirement, sort provides several command line options for sorting data in a text file.

Syntax:

$ sort [-options]

-n -If we want to sort on numeric value, then we can use -n or –numeric-sort option.

-h -If we want to sort on human readable numbers (e.g., 2K 1M 1G), then we can use -h or –human-numeric-sort option.

-M -If we want to sort in the order of months of year, then we can use -M or –month-sort option.

-c -If we want to check data in text file is sorted or not, then we can use -c or –check, –check=diagnose-first option.

24. **Find:** It use to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions.

Scanned with CamScanner

Syntax :

$ find [where to start searching from]

[expression determines what to find] [-options] [what to find]

Example:

$ find ./GFG -name sample.txt

It will search for sample.txt in GFG directory.

25. **Man:** man command in Linux is used to display the user manual of any command that we can run on the terminal.

Syntax :

$man [OPTION]... [COMMAND NAME]...

2. Write a script to find the complete path for any file

```
clear
echo "Enter File Name : \c "
read fileName
if [ -f $fileName ]
then
str=`find $fileName`
path=`pwd`
echo "Full path of file is $path/$str"
else
echo "file [ $fileName ] not exist in \c "
pwd
fi
```

2. Write a script to find the complete path for any file.

Program:

```
echo "Enter any file or file path"
Read file
Ls – l $file
```

Output:

```
Enter any file name or file path
Cut
-rw-r- -r- -  1 user 0 2013-08-27 11:15 a.ls
```


Scanned with CamScanner

3. write a shell script to execute following commands.

1. sort file abc.txt and save this sorted file in xyz.txt
2. give an example of : to execute commands together without affecting result to each other .
3. how to print "this is
   a three –line
   1.Text message"
4. which command display version of the UNIX?
5. how would u get online help of cat **command?**

Program:

```
echo "1. Sort file abc.txt and save this sorted file in xyz.txt"
echo "2. give an example of : to execute commands together without affecting result
to    each other ."
echo "3. how to print "this is
                    a three –line
                      Text message"
echo "4. which command display version of the UNIX?"
echo "5. how would u get online help of cat command?"

echo "Enter your choice :="
read ch
case Sch in
 1)sort abc.txt >> xyz.txt;;
 2)ls –l ; ls; ;
 3)printf "this is
        >a three - line
        >\t  1. Text message";;
 4)uname –a;;
 5)man cat;;
 *)echo "Invalid  option"
Esac
```

Output:

1.sort file abc.txt and save this sorted file in xyz.txt

Scanned with CamScanner

2.give an example of : to execute commands together without affecting result to each other .

3.how to print "this is

a three –line

Text message"

4. which command display version of the UNIX?

5. how would u get online help of cat command?

Enter your choice :=

4

Linux ubantu 2.6.31-14-generic #48-ubantu SMP Fri Oct 16 14:04:26 UTC 2009 i686

GNU/Linux

4. write a shell script to execute following commands:

1. how would u display the hidden files?
2. how delete directory with files?
3. how would user can do interactive copying?
4. how would user can do interactive deletion of files?
5. explain two functionality of "mv" command with example?

Program:

```
echo "1. how would u display the hidden files?"
echo "2.how delete directory with files?"
echo "3.how would user can do interactive copying?"
echo "4.how would user can do interactive deletion of files?"
echo " 5.explain two functionality of "mv" command with example?"

echo "Enter your choice :="
read ch

case $ch in
        1)ls .[a-z]*;;
        2)rm -rfabc;;
        3)cp -i abc.txt pqr.txt;;
        4)rm -i abc.txt;;
        5)      echo "1.to rename of file."
```


Scanned with CamScanner

```
                echo "2.to move a group of file to different directory."
                echo "Enter your choice :="
                read c
                case $c in
                        1)mv abc.txt pqr.txt;;
                        2)mv pqr.txt abc.txt Desktop;;
                        *)echo "Invalid option"
                esac;;
        *)echo "Invalid option"
esac
```

Output:

1. how would u display the hidden files?
2. how delete directory with files?
3. how would user can do interactive copying?
4. how would user can do interactive deletion of files?
5. explain two functionality of "mv" command with example?

Enter your choice :=

1

.bash_history .bash_profile .copy.sh.swp .edit.swp .exist.sh.swp .gtkrc
.bash_logout .bashrc .editor.swp .emacs .first2.txt
.viminfo
.kde:

5. write an shell script to execute following commands:

1. create a file called text and store name,age,and address in it.
2. display the contents of the file text on the screen.
3. delete the directories mydir and newdir at one shot.
4. sort a numeric file?
5. change the permissions for the file newtext to 666.

Program:

```
echo "1.create file and store value in it"
echo "2.display contents of file"
echo "3.delete directory"
echo "4.sort a numeric file"
```


Scanned with CamScanner

```
echo "5.change thye permission"

echo "enter your choice:"
read ch

case $ch in
    1)sh scr5.txt;;
    2)cat abc.txt;;
    3)rmdirtybca;;
    4)sort -n a.sh;;
    5)chmod 666 abc.txt;;
    *)echo "invalid choice"
Esac
```

Output:

```
1. create file and store value in it
2. display contents of file
3. delete directory
4. sort a numeric file
5. change thye permission
enter your choice:
1
enter your name:
snehal
enter your age:
20
enter your address
naroda
your name is: snehal
your age is: 20
your name is: naroda
```

6. write shell script that accept filename and displays last modification time if file exists.Otherwise display appropriate message.

Program:

Scanned with CamScanner

```
echo "enter filename"
read filename
if [ -f $filename ] ; then
        echo "file exist and modification time is "; l –l $filename | cut –c 50-54
else
        echo "file not exist"
fi
```

Output:

Enter filename
Fiule1.txt
File exist and modification time is
08:05

7. write a shell script to display the login names that begin with 's'.

Program:

```
who  -H | grep [s]*
```

Output:

NAME	LINE	TIME	COMMENT
USER	sst	2013-08-7 05:58	(:0)

8. write a shell script to remove the zero sized file from the current directory.

Program:

```
for I in `ls`
do
 if [ -s $i ]
 then
        echo "$i - Non-empty file"
else
        echo "$i – Empty file deleted"
        rm $i
 fi
done
```

Output:

04 - Non-empty file

Scanned with CamScanner

18 - Non-empty file
a - Empty file deleted
b - Empty file deleted
c - Empty file deleted

9. write a shell script to display the name of all the executable file from the current directory.

Program:

```
for File in *
    do
    if [ -x $File ]
    then
        echo $File
    fi
    done
```

Output:

```
mydir
    temp
tempdir
```

10. write a shell script that will display welcome message according to time.

Program:

```
check-`date +%H`
if [ $check -ge 06 -a $check -le 12 ]
then
  echo "Good morning"
elif [ $check -ge 12 -a #check -le 17 ]
then
  echo "Good afternoon"
else
  echo "Good evening"
fi
```


Scanned with CamScanner

Output:

Good afternoon

11. write a shell script to find number of ordinary files and directory files.

Program:

```
f=0
d=o
for i in `ls -l|tr -s " "|cut -c 1`
do
 if [ $i = "_" ]
 then
        f=`expr $f + 1`

elif [ $i = "d" ]
then
        d=`expr $d + 1`
fi
done
echo "No of ordinary files are $f"
echo "No of directories are $d"
```

Output:

No of ordinary files are 50

No of directories are 88

12. write a shell script that takes a filename from the command line and checks whether the file is an ordinary file or not.

- **If it is an ordinary file then it should display the contents of the file.**
- **If it is not an ordinary file then script should display the message: "file does not exist or is not ordinary,cannot display."**

Program:

```
echo "enter file name:"
read a

if [ -f $a ]; then
```


Scanned with CamScanner

```
        echo "ordinary file"$a
        cat $a;
else
        echo "file does not exist or ordinary file,cannot display"
fi
```

Output:

```
enter file name:
scr10.sh
ordinary file  scr10.sh
```

13. Write a shell script that takes a filename from the user and checks whether it is a directory file or not.

- **If it is a directory, then the script should display the contents of the directory.**
- **If it is not a directory file then script should display the message: "File is not a directory file."**

Program:

```
echo "enter file name:"
read a
if [ -d $a ]; then
      echo "directory : "$a
      ls $a
else
      echo "not directory,it is ordinary file"
fi
```

Output:

```
enter file name:
snehal
directory : snehal
data.txt  new-data1.txt  new-data.txt  newdata.txt  new.sh  new.txt  n.txt  pro1.sh
stu1.txt  stu.sh  t1.txt
```

14. Write a shell script that takes a filename as an argument and checks if the file exists and is executable.

1. **If the file is executable then the shell script should display the message:**

Scanned with CamScanner

"File exists"

2. **If the file does not exist and is not executable then the script should display the message: "File does not exist or is not executable."**

Program:

```
echo "enter file: "
read a
if [ -x $a ]; then
      echo "file is executable : "$a
      cat $a
else
      echo "file is not executable : "
fi
```

Output:

```
enter file:
scr10.sh
file is executable : scr10.sh
```

15. write a shell script that displays all subdirectories in current working directory.

Program:

```
Ls -F1 | grep /
```

Output:

```
desktop/
kksica/
college/
```

16. write a shell script calculates the number of ordinary and directory files in your current working directory.

Program:

```
f=0
d=0
for I in `ls -l|tr -s " "|cut -c 1`
do
 if [ $i = "_" ]
 then
```


Scanned with CamScanner

```
        f=`expr $f + 1`
elif [ $i = "d" ]
then
        d=`expr $d + 1`
fi
done
echo "No of ordinary files are $f"
echo "No of directories are $d"
```

Output:

No of ordinary files are 40

No of directories are 80

17. write a shell script that accepts 2 filenames and checks if both exists; if both exist then append the content of the second file into the first file.

Program:

```
echo -e "Enter filename1: \c"
read f1
echo -e "Enter filename2: \c"
read f2
if [ -s $f1 -a -s $f2 ]
then
  cat $f2 >>$f1
else
echo "Cannot append"
fi
```

Output:

```
Enter filename1: f1
Enter filename2: f2
user@ubuntu:~$cat f1
HIHELLP
kksica
Welcome
```


Scanned with CamScanner

18. Write a shell script that takes the name of two files as arguments and performs the following:

i. Displays the message :
"Displaying the contents of file :(first argument)"
and displays the contents page wise.

ii. Copies the contents of the first argument to second argument.

iii. Finally displays the message :
"File copied successfully."

Program:

```
echo 1.display the content of two files
echo 2.copy the files
echo 3.display the message
echo enter your choice
read ch

case $ch in
      1)echo enter file1
       read a
       echo enter file 2
       read b
       cat $a + $b;;
      2)echo enter file1
       read c
       echo enter file for copy
       read d
       if [ -f $c -a -f $d ]; then
           cp $c $d
           echo file copied successfully
       else
           echo not copied
       fi;;
esac
```

Output:

Scanned with CamScanner

```
1. display the content of two files
2. copy the files
3. diplay the message
enter your choice
1
enter file1
scr9.sh
enter file 2
scr10.sh
for i in *
do
  if [ -x $i ]; then
          echo "executable"$i
  else
          echo "not executable"
  fi
done
a=`date +%H`
if [ $a -gt 7 -a $a -le 12 ]; then
      echo "good morning"
elif [ $a -gt 12 -a $a -le 4 ]; then
      echo "good afternoon"
elif [ $a -gt 4 -a $a -le 8 ]; then
      echo "good evening"
elif [ $a -gt 8 -a $a -le 1 ]; then
  echo "good night"
else
  echo "invalid time"
fi
```

19. Write a shell script to display the following menu and acts accordingly:

1. Calender of the current month and year.

2. Display "Good Morning/Good Afternoon/Good Evening" according to the current login time.

Scanned with CamScanner

3. User name,user home directory.

4. Terminal name, Terminal type.

5. Machine name.

6. No. of users who are currently logged in; List of users who are currently logged in.

Program:

```
echo 1.calemder of current month and year
echo 2.display msg according to time
echo 3.use rname, user home directory
echo 4.terminal name,terminal type
echo 5.machine name
echo 6.no. of users who are cuurently login
echo enter your choice
read ch

case Sch in
      1)cal 8 2015;;
      2)a=`date +%H`
            if [ Sa -gt 7 -a Sa -le 12 ]; then
                  echo "good morning"
elif [ Sa -gt 12 -a Sa -le 4 ]; then
                  echo "good afternoon"
elif [ Sa -gt 4 -a Sa -le 8 ]; then
                  echo "good evening"
elif [ Sa -gt 8 -a Sa -le 1 ]; then
                  echo "good night"
            else
                  echo "invalid time"
            fi
            ;;
      3)uname -a;;
      4)tty;;
      5);;
      6)who -H;;
```


Scanned with CamScanner

```
esac
```

Output:

```
1. calemder of current month and year
2. display msg according to time
3. username user home directory
4. terminal name,terminal type
5. machine name
6. no. of users who are cuurently login
enter your choice
1
   August 2015
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
```

20. write a shell script that displays the following menu and scts accordingly:

1. Concatenates two strings
2. Renames a file
3. Deletes a file.
4. Copy the file to specific location

Program:

```
echo 1.concate the two string
echo 2.rename the file
echo 3.deletes a file
echo 4.copy file to the  specific location
echo enter your choice:
read ch
case Sch in
      1)echo enter the first string:
```


Scanned with CamScanner

```
    read a
    echo enter the second string:
    read b
    echo $a$b ;;
  2)echo enter the file name for rename:
    read a
    echo enter the new file name:
    read b
    mv $a $b;;
  3)echo enter the file name for delete:
    read a
    rm $a;;
  4)echo enter file for copy:
    read a
    echo enter the location where you want to copy:
    read b
    cp $a $b;;
esac
```

Output:

```
1.concate the two string
2.rename the file
3.deletes a file
4.copy file to the specific location
enter your choice:
1
enter the first string:
sne
enter the second string:
hal
snehal
```

21. Write a shell script to change the suffix of all your *.txt files to .dat.

Program:

Scanned with CamScanner

```
for i in `ls *.txt`
do
    f=`echo $i | cut -d "." -f1`
    mv $i $f.dat
done
```

Output:

```
[user@localhost ~]$ ls
1      a.txt   Documents o.txt       palindrom.sh practical.sh scr10.sh scr14.sh
scr17.sh   scr19.sh scr3_1.sh scr6.sh   scr8.sh    Templates     xyz.txt
ABC.sh b.sh    Download p4.sh       Pictures    prac.txt    scr11.sh scr15.sh
scr18_1.sh scr20.sh scr4.sh   scr7_1.sh scr9.sh     untitled folder
abc.txt demo.shsnehal     p5.sh       prac1.txt   pro3.sh     scr12.sh scr16.sh
scr18_2.sh scr21.sh scr5.sh   scr7.sh   script3_1.sh Videos
nikita Desktop Music    palindrome.sh prac.sh~    Public      scr13.sh scr17_1.sh
scr18.sh   scr2.sh  scr5.txt  scr_7.txt s.txt       xyz.tx

[user@localhost ~]$ ls
1      nikita  Documents o.datpalindrom.sh prac.sh~    scr10.sh scr14.sh   scr17.sh
scr19.sh scr3_1.sh scr6.sh   scr8.sh    Templates     xyz.tx
abc.dat b.sh   Download p4.sh       Pictures    practical.sh scr11.sh scr15.sh
scr18_1.sh scr20.sh scr4.sh   scr7_1.sh scr9.sh     untitled folder
ABC.shdemo.shsnehal     p5.sh       prac1.dat   pro3.sh     scr12.sh scr16.sh
scr18_2.sh scr21.sh scr5.dat  scr_7.dat script3_1.sh Videos
a.dat  Desktop Music    palindrome.sh prac.dat    Public      scr13.sh scr17_1.sh
scr18.sh   scr2.sh  scr5.sh   scr7.sh    s.dat       xyz.dat
```

22. write a shell script to accept a directory- name and display its contents. If input is notGiven then HOME directory's contents should be listed. (Make use of command line argument).

Program:

```
if [ $1 –eq 0 ]
then
  ls –l home
else
  ls -l $1
```


Scanned with CamScanner

```
fi
```

Output:

```
user@ubuntu:~$ sh 22.sh kksica
-rw-r-- r--  1 user 0 2.13-08-27  17:06 file1
```

23. Write a shell script to get all files of home directory and rename them if their names Start with c.

Newname = oldname111

Program:

```
cd
y="111"
c="c"
for I in `ls`
do
        if [ -d Si ]
        then
                x=`echo Si| cut -c 1`
                if [ Sx = Sc ]
                then
                mv -f Si  Si Sy
        fi
done
```

24. Write a shell script that takes two filename as arguments. It should check whether the Contents of two files are same or not, if they are same then second file should be delete.

Program:

```
echo "Enter first file name "
read file1
echo "Enter second file name "
read file2
cmp $file1 $file2> error
total=`wc  -c error | cut –f 7 –d " "`
echo $total
```


Scanned with CamScanner

```
if [ $total -eq 0 ]
then
  echo "Both file's contents are same"
else
  echo "Both file's contents are not same"
fi
```

Output:

```
Enter first file name
File1
Enter second file name
File2
Both file's contents are same
```

25. Write a shell script that accepts two directory names from the command line and copies all the files of one directory to another. The script should do the following:

- **If the source directory does not exist, flash a error message**
- **If destination directory does not exist create it.**
- **Once both exist copy all the files from source directory to destination directory.**

Program:

```
echo "Enter source folder name:"
read file1
if [ -d "$file1" ]; then
  echo "folder $file1 exist."
  else
  echo "can not find source folder $file1"
  fi
echo "Enter destination filder name:"
  read file2

if [ -d "$file2" ]; then
```


Scanned with CamScanner

```
  echo "folder $file2 exist."
  cp * "$file1" "$file2"
  echo "copies succefully.."
else
  mkdir "$file2"
  cp * "$file1" "$file2"
  echo "copies succefully.."
fi
```

Output:

```
Enter source folder name:
abc
folder abc exist.
Enter destination filder name:
xyz
cp: -r not specified; omitting directory 'a'
cp: -r not specified; omitting directory 'abc'
cp: -r not specified; omitting directory 'neha'
cp: -r not specified; omitting directory 'sonu'
cp: -r not specified; omitting directory 'unit2'
cp: -r not specified; omitting directory 'xyz'
cp: -r not specified; omitting directory 'abc'
copies succefully..
```

26. Write a shell script that displays the following menu:

- **List home directory**
- **Date**
- **Print working directory**
- **Users logged in**

Read the proper choice. Execute corresponding command. Check for invalid choice.

Program:

```
echo "1. List home directory "
echo "2.Date "
echo"3.Print working directory "
```


Scanned with CamScanner

```
echo"4.Users logged in "
echo "Enter Your Choice := "
read ch
case $ch in
1) Ls/home;;
2) Date +"%h %m %d";;
3) Pwd;;
4) Who;;
*)echo "Invalid Option"
esac
```

Output:

```
1.      List home directory
2.      Date
3.      Print working directory
4.      Users logged in

Enter your choice :=
2
Aug 08 28
```

27. Write a shell script that displays all hidden files in current directory.

Program:

```
echo "Enter source folder name:"
read file1
if [ -d "$file1" ]; then
  echo "folder $file1 exist."
  ls -la "$file1"
else
  echo "can not find source folder $file1"
fi
```

Output:

```
Enter source folder name:
```


Scanned with CamScanner

```
Abc
folder abc exist.
total 44
drwxr-xr-x 2 user user 4096 Feb 10 18:13 .
drwxr-xr-x 8 user user 4096 Feb 25 18:32 ..
-rwxr-xr-x 1 user user  138 Feb 10 18:18 b.sh
-rw-r--r-- 1 user user   22 Dec 17 17:09 hello
-rwxr-xr-x 1 user user  165 Feb 10 18:18 p1.sh
-rwxr-xr-x 1 user user  139 Feb 10 18:18 p2.sh
-rwxr-xr-x 1 user user  363 Feb 10 18:18 pro25.sh
-rw-r--r-- 1 user user  127 Feb 10 18:18 u2_p2.sh
-rwxr-xr-x 1 user user  298 Feb 10 18:18 u2p2.sh
-rwxr-xr-x 1 user user  164 Feb 10 18:18 u2p4.sh
-rwxr-xr-x 1 user user  284 Feb 10 18:18 xyz.sh
```

28. Write a shell script that combine two files in the third file horizontally and vertically.

Program:

```
echo "Enter Filename1: "
read f1
echo "Enter Filename2: "
read f2
cat $f1>vfile
cat $f2 >>vfile
paste -d " " $f1 $f2 >>vfile
```

Output:

```
Enter Filename1:
f1
Enter Filename2:
F2
user@ubuntu:~$cat vfile
HIHELLP
ASDSAD
Welcome
```


Scanned with CamScanner

HI HELLP welcome

ASDSAD

29. Write a shell script to delete all the spaces from a given file.

Program:

```
Echo  "Enter the Filename:  "
read fname
cat  $fname  | tr  –d  "  "  > $fna
```

Output:

```
Enter the Filename:
f1
user@ubuntu:~$cat   f1
HIHELLP
KKSICA
```

30. Write a shell script to find a given date fall on a weekday or a weekend.

Program:

```
if  [  $#  -eq  2  ]
then
  x=`cat $2 07  | wc -1 `
        x=`expr  $x  - 1 `
        cal  $2  07  | tail  -$x  > file
        cat file  |  grep  -w  "$1"  >  f
        j=0
        for  i  in  `cat  f
        do
          j=`expr   $j  + 1 `
          if  [  $i  = $1  ]
        then
          break
        fi
  done
if  [  $j  -gt  1  -a  $j  -lt  7  ]
then
```


Scanned with CamScanner

```
  echo "Day is Weekday"
 else
  echo"Day is Weekend"
 fi

 else
  echo "Invalid arguments"
 echo "Usage : 10 day month"
 fi
```

Output:

```
user@ubuntu:-$ sh 10 11 07
Day is weekday
```

31. Write a shell script to search for a given word in all the files given as the argument on the command line.

Program:

```
If [ $# -lt 1 ]
Then
 echo "Invalid argument"
else
 echo "Enter a word: "
 read word
 x=0
for i in `echo $*`
do
 if [ -f $i ]
 then
 x=`cat $i | grep –c "$word"`
 if [ $x –ne 0 ]
 then
        echo "word found in files : $i"
        break
 fi
fi
```


Scanned with CamScanner

```
done
fi
```

Output:

```
user@ubantu:~$ sh 11 f1 f2 file
Enter a word:
Su
Word found in file : file
```

32. Write a shell script that display last modified file in the current directory.

Program:

```
echo "Enter filename"
read filename
if [ -f $filename ] ; then
  echo "file exist and modification time is "; ls –l $filename | cut –c 50-54
else
  echo "file not exist"
fi
```

Output:

```
Enter filename
File1.txt
File exist and modification time is
08:05
user@ubuntu:~$
```

33. Write a script to display the permissions of the particular file.

Program:

```
Ls –l
```

Output:

```
-rw-r—r-- 1 root root 2453 jul 17 16:25 /etc/passwd
```

34. Write a shell script to display the calendar in the following manner:

i. Display the calendar of months m1 and m2 by 'CAL m1 , m2' command file.

ii. Display the calendar of the months from m1 to m2 by 'CAL m1-m2' command file.

Scanned with CamScanner

Program:

```
If [ $# -eq 1 ]
Then
  y= "_"
     z= ","
  l=`echo $1 | wc -c`
  i=1
  c=0
  while [ $i -le $l ]
  do
          x=`echo $1 | cut -c $i`
          if [ $x = $y ]
          then
                  c=1
                  break
          elif [ $x = $z ]
          then
                  c=2
                  break
          fi
          i=`expr $i + 1`
  done
  if [ $c -eq 1 ]
  then
          first=`echo $1 | cut -d "_" -f 1`
          second=`echo $1 | cut -d "_" -f 2`
          while [ $first -le $second ]
          do
                  cal $first 07
                  first=`expr $first +1`
          done
  elif [ $c -eq 2 ]
  then
          first=`echo $1 | cut -d "," -f 1`
```


Scanned with CamScanner

```
        second=`echo $1 | cut -d "," -f 2`
        cal $first 07
  else
        echo "Argument not entered correctly"
  fi
else
      echo "Invalid argument usage: 12 m1,m2 or 12 m1-m2"
fi
```

Output:

```
user@ubantu:~$ sh12 5,7

        May 7
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

        July 7
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
user@ubantu:~$ sh12 6-7
        July 7
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
```


Scanned with CamScanner

26 27 28 29 30

July 7

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						

35. Write a shell script to display the following for a particular file:

i. Display all the words of a file in ascending order.

ii. Display a file in descending order.

iii. Toggle all the characters in the file.

Iv. Display type of the files.

Program:

```
echo "Enter the filename: "
read fn
echo "i.Display all the words of a file in ascending order."
echo "ii.Display a file in descending order."
echo "iii.Toggle all the characters in the file."
echo "iv. Display type of the files."
echo "v. Display type of the file."
echo "Enter your choice:"
read c

case $c in
i)for I in `cat $fn`
  do
  echo $i>>ff
  done
  echo " File in ascending order:"
  cat ff | sort
```


Scanned with CamScanner

```
  rm -f ff;;
ii)for I in `cat $fn`
  do
  echo $i>>ff
  done
  echo " File in decending order:"
  cat ff | sort -r
   rm -f ff;;
iii)x=`cat $fn | wc -l`
i=1
   while [ $1 -le $x ]
   do
  s=`cat $fn | head -$i | tail -1`
  xx=`echo $s | wc -c`
  j=1
  str=""
  while [ $j -le $xx ]
  do
          c=`echo $s | cut -c $j`
          case $c in
          [A-Z])c=`echo $c | tr [A-Z] [a-z]`;;
          [a-z])c=`echo $c  tr [a-z] [A-Z]`;;
          esac
          str=$str$c
          j=`expr $j + 1`
  done
  echo $str >> ff

  i=`expr $i + 1`
   done;;
iv)if [ -f $fn ]
  then
  echo "$fn is ordinary file"
elif [ -d $fn ]
```


Scanned with CamScanner

```
then
 echo "$fn is directory file"
elif [ -x $fn ]
 then
 echo "$fn is executable file"
elif [ -r $fn ]
 then
 echo "$fn is Readable file"
elif [ -w $fn ]
 then echo "$fn is writable file"
fi;;
esac
```

Output:

```
Enter the filename:
File
i.Display all the words of a file in ascending order.
ii.Display a file in descending order.
iii.Toggle all the characters in the file.
Iv.Display type of the files.
Enter your choice:i
file in ascending order:
asdjlasjdsd;jas
askdjaskljdas
is
thdjasdjlasd
unix
```

36. Write a shell script to check whether the named user is currently logged in or not.

Program:

```
echo "Enter the user name :\c"
read name
>>temp
echo "The user \c " $name
```


Scanned with CamScanner

(who | grep $name >> temp) && echo " Is currently logged in " || echo " is not logged in"

Output:

Enter the user name:

Pts/1

Is currently logged in

37. write a shell script to display the following menu for a particular files:

i.display all the words of a file in ascending order.

ii.display a file in descending order.

iii.display a file in reerse order.

iv.toggle all the characters in the file

v.display type of the file.

Program:

```
echo  "Enter he filename: "
read fn
echo "i.display all the words of a file in ascending order."
echo "ii.display a file in descending order."
echo "iii.display a file in reerse order."
echo "iv.toggle all the characters in the file"
echo "v.display type of the file."
echo "Enter your choice:"
read c

case  $c  in
i)for  I  in  `cat  $fn`
  do
  echo  $i>>ff
  done
  echo "File in ascending order: "
  cat  ff | sort
  rm  -f  ff;;
```


Scanned with CamScanner

```
            esac
            str=$strSc
            j=`expr $j + 1`
    done
    echo $str >> ff

    i=`expr $i + 1`
    done;;
v)if [ -f $fn ]
   then
   echo "$fn is ordinary file"
elif [ -d $fn ]
   then
   echo "$fn is direcory file"
elif [ -x $fn ]
   then
   echo "$fn is Executable file"
elif [ -r $fn ]
   then
   echo "$fn id Readable file"
elif [ -w $fn ]
   then echo "$fn is writable file"
   fi;;
esac
```

Output:

```
Enter the filename:
File
i.display all the words of a file in ascending order.
ii.display a file in descending order.
iii.display a file in reerse order.
iv.toggle all the characters in the file
v.display type of the file.
Enter your choice:
i
```


Scanned with CamScanner

```
ii)for I in `cat $fn`
  do
  echo $i>>ff
  done
   echo "File in decending order: "
   cat ff | sort -r
   rm -f ff;;
iii)for i in `cat $ fn`
   do
  echo $i>>ff
   done
   x=`cat ff | wc -l`
   echo "Reverse order:"
   while [ $x -gt 0 ]
   do
  s=`cat ff | head -$x | tail -1`
  echo $s
  x=`expr $x -1`
  done
  rm -f ff;;
iv)x=`cat $fn | wc -l`
i=1
  while [ $i -le $x ]
  do
  s=`cat $fn | head -$i | tail -1`
  xx=`echo $s | wc -c
  j=1
  str=" "
  while [ $j -le $xx ]
  do
        c=`echo $s | cut -c $j`
        case $c in
        [A-Z])c=`echo $c | tr [A-Z] [a-z]`;;
        [a-z])c=`echo $c | tr [a-z] [A-Z]`;;
```


Scanned with CamScanner

```
            esac
            str=$str$c
            j=`expr $j + 1`
    done
    echo $str >> ff

    i=`expr $i + 1`
    done;;
v)if [ -f $fn ]
    then
    echo "$fn is ordinary file"
elif [ -d $fn ]
    then
    echo "$fn is direcory file"
elif [ -x $fn ]
    then
    echo "$fn is Executable file"
elif [ -r $fn ]
    then
    echo "$fn id Readable file"
elif [ -w $fn ]
    then echo "$fn is writable file"
    fi;;
esac
```

Output:

```
Enter the filename:
File
i.display all the words of a file in ascending order.
ii.display a file in descending order.
iii.display a file in reerse order.
iv.toggle all the characters in the file
v.display type of the file.
Enter your choice:
i
```


Scanned with CamScanner

file in ascending order:
asdjlasjdsd;jas
askdjaskljdas
is
thdjasdjlasd
unix

38. Write a shell script to find total no. of users and finds out how many of them are currently logged in.

Program:

```
total=`cat /etc/passwd | wc -l`
cur_log=`who | wc -l`

echo "Total users        : $total"
echo "Currently logged: $cur_log"
```

Output:

Total users :32
Currently logged:3

39. Write a shell script that displays the directory information in the following format-

Filename size Date protection Owner

Program:

```
x=`ls -l | wc -l`
i=2
echo -e "Filename\t\tsize\tdate\tProtection\tQwner\n"
while [ $i -le $x ]
do
 s=`;s -l | head -$i | tail -` | tr -s " "`
 fn=`echo $s | cut -d " " -f 9`
 si=`echo $s | cut -d " " -f 5`
      d1=`echo $s | cut -d " " -f 6`
d2=`echo $s | cut -d " " -f 7`
p=`echo $s | cut -d " " -f 1`
```


Scanned with CamScanner

```
o=`echo $s | cut -d " " -f 3`
echo -e " $fn\t\t\t$si\t$d1 $d2\t$p\t$o"
i=`expr $i + 1`
done
```

Output:

Filename	size	Date	protection	Owner
01	2	2013-08-27	-rw-r--r--	s112
02	131	2013-08-6	-rw-r--r--	s112
03	0	2013-08-7	-rw-r--r--	s112
04	128	2013-08-4	-rw-r--r--	s112
05	73	2013-08-11	-rw-r--r--	s112
06	213	2013-08-11	-rw-r--r--	s112
07	72	2013-08-6	-rw-r--r--	s112
08	299	2013-08-18	-rw-r--r--	s112
09	70	2013-08-11	-rw-r--r--	s112
10	363	2013-08-5	-rw-r--r--	s112
11	238	2013-08-6	-rw-r--r--	s112
12	254	2013-08-18	-rw-r--r--	s112
13	129	2013-08-6	-rw-r--r--	s112
14	278	2013-08-5	-rw-r--r--	s112
15	488	2013-08-25	-rw-r--r--	s112
16	1429	2013-08-6	-rw-r--r--	s112
17	665	2013-08-27	-rw-r--r--	s112
18	475	2013-08-6	-rw-r--r--	s112
19	323	2013-08-29	-rw-r--r--	s112
20	133	2013-08-5	-rw-r--r--	s112

40. Write a shell script to display five largest files from the current directory.

Program:

```
Ls -S | head -5
```

Output:

Desktop

Scanned with CamScanner

Document
Download
Music
Pictures

41. Write a shell script that toggles contents of the file.

Program:

```
echo "Enter the filename: "
 read fn
x=`cat $fn | wc -l`
 i=1
 while [ $i -le $x ]
 do
        s=`cat $fn | head -$i | tail -1`
        xx=`echo $s | cut -s $j`
        j=1
        str=" "
        while [ $j -le $xx ]
        do
                c=`echo $s | cut -c $j`
                case $c in
                [A-Z])c=`echo $c | tr [A-Z] [a-z]`;;
                [a-z])c=`echo $c | tr [a-z] [A-Z]`;;
                esac
                str=$str$c
                j=`expr $j + 1`
        done
        echo $str >> ff

        i=`expr $i + 1`
done;;
```

Output:

Scanned with CamScanner

Enter the filename:

Filel

Kksica

42. Write a shell script that report whether your friend has currently logged in or not. If he has logged in then the shell script should send a message to his terminal suggesting a dinner tonight. If you do have write permission to his terminal or if he hasn't logged in then such a message should be mailed to him about your dinner proposal.

43. Write a shell script for the performing the write and mail.

Program:

```
TO_ADDRESS=recipient@domain.com
FROM_ADDRESS="sender"
SUBJECT="Mail server hosting demo"
BODY="This is a linux mail system.Linux is one of the email operating systems which can be used to send and receive emails."
echo ${BODY}| mail -s ${SUBJECT} ${TO_ADDRESS} -- -r{FROM_ADRESS}
echo "Mail is successfully send"
```

44. Write a shell script to accept any character using command line and list all the files starting with that character in the current directory.

Program:

```
echo "$1"
ls [$1]
```

Output:

```
$sh script44.sh h
Hello hii.sh hi.sh
```

45. Create a file called student containing roll-no, name and marks. a. Display the contents of the file sorted by marks in descendingorder b. Display the names of students in alphabetical order ignoring thecase. c. Display students according to their rollnos. d. Sort file

Scanned with CamScanner

according to the second field and save it to file'names'. e. Display the list of students who scored between 70 and 80.

Program:

```
echo "1.Display the content of the file sorted by marks in Desending order"
echo "2.Display the names of students in alphabetical order ignoring the case"
echo "3.Display student according to their roll nos"
echo "4.sort file according to the second field and save it to 'names"
echo "5.Display the list of student who scored between 70 and 80 "
echo "Enter your choice :"
read ch
case $ch in
1)cat student | sort -r -t "|" -k3;;
2)cat student | cut -d "|"-f2 | sort -i;;
3)cat student | cut -d "|" -f1,2 | sort -t "|" -k1;;
4)cat student | sort -t "|" -k2 > names;;
5)m1=` sed -n '1p' student | cut -d "|" -f3 `
 m2=` sed -n '2p' student | cut -d "|" -f3 `
 m3=` sed -n '3p' student | cut -d "|" -f3 `
 m4=` sed -n '4p' student | cut -d "|" -f3 `
 m5=` sed -n '5p' student | cut -d "|" -f3 `

 if [ $m1 -ge 70 -a $m1 -le 80 ]
 then
sed -n '1p' student
 fi
 if [ $m2 -ge 70 -a $m2 -le 80 ]
 then
sed -n '2p' student
 fi
 if [ $m3 -ge 70 -a $m3 -le 80 ]
 then
sed -n '3p' student
 fi
 if [ $m4 -ge 70 -a $m4 -le 80 ]
```


Scanned with CamScanner

```
  then
sed -n '4p' student
  fi
  if [ $m5 -ge 70 -a $m5 -le 80 ]
  then
sed -n '5p' student
  fi;;
esac
```

Output:

1. Display the content of the file sorted by marks in Desending order
2. Display the names of students in alphabetical order ignoring the case
3. Display student according to their roll nos
4. Sort file according to the second field and save it to 'names
5. Display the list of student who scored between 70 and 80

Enter your choice :

1

Roll no:05

Roll no:04

Roll no:03

Roll no:02

Roll no:01

Name:urvisha

Name:tejal

Name:sonu

Name:rajvi

Name:Minali

Marks:96

Marks:79

Marks:70

Marks:60

Marks:60

☺ ☺ ☺ ☺ ☺

Scanned with CamScanner