Bapuji Educational Association(Regd.)

Bapuji Institute of Engineering and Technology, Davangere-577 004

Department of Master of Computer Applications

Semester : I

Unix Programming Lab Manual

[20MCA17]

By : Dr.Shankaragowda B.B. MCA., M.Phil., Ph.D., MISTE
Assistant Professor,
Dept. of MCA,
Bapuji Institute of Engineering & Technology,
Davangere-577 004.

CONTENTS

.A. Explore the Unix environment

B. Explore vi editor with vim tutor.

C. Shell Programming

6	a)Write a shell script to find a file/s that matches a pattern given as command line argument in the home directory, display the contents of the file and copy the file into the directory ~/mydir	15
	b) Write a shell script to list all the files in a directory whose filename is at least 10 characters. (use expr command to check the length)	
7	a)Write a shell script that gets executed displays the message either "Good Morning" or "Good Afternoon" or "Good Evening" depending upon time at which the user logs in.	16
	b)Write a shell script that accept a list of filenames as its argument, count and report occurrence of each word that is present in the first argument file on other argument files.	
8	a)Write a shell script that determine the period for which a specified user is working on system and display appropriate message.	17
	b)Write a shell script that reports the logging in of a specified user within one minute after he/she log in. The script automatically terminate if specified user does not log in during a specified period of time.	
9	a)Write a shell script that accept the file name, starting and ending line number as an argument and display all the lines between the given line number.	18
	b) Write a shell script that folds long lines into 40 columns. Thus any line that exceeds 40 characters must be broken after 40th, a "\" is to be appended as the indication of folding and the processing is to be continued with the residue. The input is to be supplied through a text file created by the user.	
10	a) Write an awk script that accepts date argument in the form of dd-mm-yy and displays it in the form if month, day and year. The script should check the validity of the argument and in the case of error, display a suitable message.	20
	b) Write an awk script to delete duplicated line from a text file. The order of the original lines must remain unchanged.	
11	a) Write an awk script to find out total number of books sold in each discipline as well as total book sold using associate array down table as given below. Electrical 34 Mechanical 67 Electrical 80 Computer Science 43 Mechanical 65 Civil 98 Computer Science 64	21
	b) Write an awk script to compute gross salary of an employee accordingly to rule given below. If basic salary is < 10000 then HRA=15% of basic & DA=45% of basic If basic salary is >=10000 then HRA=20% of basic & DA=50% of basic.	

D. Model Viva Voce Questions

A. Explore the Unix environment

What is Unix?

Unix is an operating system. All computers have operating systems. An operating system is a software that acts as an interface between the user and the computer hardware. An operating system acts as a resources manager. Here resources mean hardware resources like the processor, the main memory, the hard disk, I/O devices and other peripherals. In addition to being a multi-user operating system, Unix gives its users, the feeling of working on an independent computer system.

Linux is an operating system that evolved from a kernel created by Linus Torvalds when he was a student at the University of Helsinki. Generally, it is obvious to most people what Linux is. However, both for political and practical reasons, it needs to be explained further. To say that Linux is an operating system means that it's meant to be used as an alternative to other operating systems, Windows, Mac OS, MS-DOS, Solaris and others. Linux is not a program like a word processor and is not a set of programs like an office suite. Linux is an interface between computer/server hardware, and the programs which run on it.

Salient Features of Unix

Unix is a multi-tasking operating system – has the ability to support concurrent execution of two or more active processes. Here it may be noted that an instance of a program in execution is known as a process.

Unix is a multi-user operating system- has the ability to support more than one user to login into the system simultaneously and execute programs. For this, the Unix presents a virtual computer to every user by creating simulated processors, multiple address spaces and the like.

Unix operating system is highly portable. Compared to other OS,it is very easy to port Unix on to different hardware platforms with minimal or no modifications at all.

Unix Components:

1. The Kernel
2. The shell
3. The file system

The Kernel: The kernel is the heart of any Unix operating system. This kernel is relatively a small piece of code that is embedded on the hardware. Actually it is a collection of programs that are mostly written in c. Every Unix system has a kernel (just one) that gets automatically loaded on to the memory as soon as the system is booted. AS the kernel sits on the hardware it can directly communicate with the hardware.

The Shell: Every Unix system has, at least, one shell. A shell is a program that sits on the kernel and acts as an agent or interface between the users and the kernel and hence the hardware. It is similar to the command.com in the MS-DOS environment.

Types of Shells:
The Bourne Shell(sh)
The CShell(csh)
The kornshell(ksh)
The Bourne-Again Shell(bash)

Basic Unix Commands

pwd 'print working directory' displays the name of the current directory
cd 'change directory' command will change the current directory to the directory specified as the argument to the command, as in *'cd /home/WWW-pages/username'*. ('cd' without specified parameter will return to your home directory.)
ls 'list files' command displays the files in a directory.
ls -l 'long list option' for listing files displays permissions, links, owner, group, file size, modification date, file name.
rm 'remove' command deletes ordinary files in a directory.
mv 'move' command moves a file from one location to another. It is also used to rename files, as in *'mv thisfile.txt thatfile.txt'*
cp 'copy' command creates a copy of a file.
chmod 'change mode' command is used to control access rights to a file or files.
mkdir 'make directory' command creates a directory or subdirectory within the current directory.
rmdir 'remove directory' command removes a directory or subdirectory. The specified

directory must be empty before it can be removed.

find 'find' command is used to locate files.

file 'file' command is used to determine the type of information in the file listed as the argument to the command, i.e. text or binary.

cat 'cat' command displays the contents of files. It is also used to concatenate files as in *"cat file1.txt file2.txt file3.txt > allfiles.txt"*.

wc 'wc' command displays a count of characters, words, and lines in a text file.

sort 'sort' command is used to sort and/or merge text files.

grep 'grep' command searches for text strings in files.

Opening and Exiting Commands

```
   vi            Invokes vi with blank editing screen (new file) in command mode.
   vi filename   Invokes vi on existing file.
   :w            Writes (saves) existing file (command mode only; <ESC> from insert
mode).
   :w filename   Writes (saves) to new file (command mode only; <ESC> from insert
mode).
   :x            Writes (saves) file and exits vi (command mode only; <ESC> from
insert mode).
   :q            Quits vi without saving (command mode only; <ESC> from insert mode).
   :q!           Quits vi without saving any changes to file (command mode only;
                 <ESC> from insert mode).
```

Movement Commands

```
   h         Move cursor left one character.
   j         Move cursor down one line.
   k         Move cursor up one line.
   l         Move cursor right one character.
   w         Move cursor forward one word.
   b         Move cursor backward one word.
   e         Move cursor to end of word.
  ^F         Move cursor forward one screenful.(Hold CTRL key and
press f)
  ^B         Move cursor back one screenful. (Hold CTRL key and press
b)
  ^D         Move cursor down half screenful. (Hold CTRL key and
press d)
  ^U         Move cursor up half screenful. (Hold CTRL key and press
u)
```

Editing Commands

```
   i         Insert mode; inserts text before current cursor
position.
   I         Insert mode; inserts text at the beginning of the line.
   a         Insert mode; append text following cursor position.
   A         Insert mode; append text at the end of the line.
   o         Open a new line below the current line and insert text.
   O         Open a new line above the current line and insert text.
   r         Replace character under cursor.
   R         Overtype mode; <ESC> terminates overtype.
   s         Substitute following text for character at cursor
position; <ESC>
             terminates
             text entry mode.
   S         Substitute text on entire line.
 <ESC>       Return to visual command mode from insert mode.
   x         Delete character at cursor position.
```

X Delete character before cursor.
dw Delete word at cursor position.
dd Delete current line.
d$ Delete from cursor to end of line.
d^ Delete from cursor to beginning of line.
c*x* Change text object at cursor positon. *x* is a cursor movement key,
commonly c (line), w (word), b (back one word), $ (to end of line), ^ (to
beginning of line).
<ESC> terminates text entry mode.
y*m* Yank (copy) text block identified by movement command *m*. (See above)
Y Yank (copy) current line.
p Put yanked text after or below cursor.
P Put yanked text before or above cursor.
. Repeat last edit.
u Undo last edit.
U Restore current line.

B. Explore vi editor with vim tutor.

Perform the following operations using vi editor, but not limited to:

1. insert character, delete character, replace character

   insert character: i
   i : insert before cursor
   I : insert at beginning of line

   Delete character : x

x	deletes the character under the cursor.
X	deletes the character to the left of your cursor.
dw	deletes from the character selected to the end of the word.
dd	deletes all the current line.
D	deletes from the current character to the end of the line.

   Replace character : r
   change word : cw

2. Save the file and continue working
   :w
3. save the file a exit the editor
   :wq
4. quit the editor
   :q
5. quit without saving the file
   :q!
6. rename a file
   mv
7. insert lines, delete lines,
   i
   dd
8. set line numbers
   Set nu
9. search for a pattern

These two commands differ only in the direction where the search takes place:

- The / command searches forwards (downwards) in the file.
- The ? command searches backwards (upwards) in the file.

. matches a single character
$ End of the file

10. move forward and backward

w: Forward one word
b: Back one word

C. Shell Programming:

Code: 13MCA17

1. a)Write a shell that takes a valid directory name as an argument and recursively descend all the subdirectories, finds the maximum length of any file in that hierarchy and writes this maximum value to the standard output.

```
for i in $*
 do
if [ -d $i ]
then
 echo "large filename size is"
echo `ls -Rl $1 | grep "^-" | tr -s ' ' | cut -d' ' -f 5,8 | sort -n
| tail -1`
else
echo "not directory"
fi
done
```

Description: Sort Sorts the lines of the specified files, typically in alphabetical order. Using the –m option it can merge sorted input files. Its syntax is: sort [<options>] [<filed specifier>] [<filename(s)>]

Cd (change [current working] directory)

……

```
Input: $sh 2a.sh enter directory name
Output:
       Max file length is 835
```

……

b) Write a shell script that accepts a path name creates all the components in that path name as directories. For example, if the script is named mpc, then command mpc a/b/c/d should create directories a, a/b, a/b/c, a/b/c/d.

```
echo "enter the pathname"
read p
i=1
j=1
len=`echo $p|wc -c`
while [ $i -le $len ]
do
x=`echo $p | cut -d / -f $j`
namelength=`echo $x|wc -c`
mkdir $x
cd $x
pwd
j=`expr $j + 1`
i=`expr $i + $namelength`
echo $g
done
```

Description: mkdir(make directory)

$mkdir directory

Creates a subdirectory called directory in the current working directory. You can only create subdirectories in a directory if you have write permission on that directory.

Pwd: Displays current working directory.

……..

```
Input: Enter the pathname
       a/b/c/d
Output: a, a/b, a/b/c, a/b/c/d
```

2. a) Write a shell script that accepts two file names as arguments, checks if the permissions for these files are identical and if the permission are identical, output common permission and otherwise output each file name followed by its permissions.

```
-Check the permissions
if [ $# -eq 0 ]                #Line1
 then
     echo "arguments not entered sorry try again "
 else
       ls -l $1 > f1           #Line5
       x=`cut -c2-10 f1`       #Line6
       echo $x
       ls -l $2 > f2
       y=`cut -c2-10 f2`
       echo $y
       echo " "
       if [ $x = $y ]
       then
       echo "permission of both files are same"
       echo $x
       else
       echo "permission are different"
       echo $x
       echo $y
  fi
 fi
```

…….

Description: In Line1 $# means total number of arguments supplied to the shell script. In line5 File attributes and permissions can be known by using the listing command ls with –l option. In line6 the cut command - splitting files vertically. Using this command required field(s) or column(s) can be extracted from a file. The –c option is used to extract required fields based on character positions or column(s).

If-then-else

The syntax of the if-then-else construct is

```
If[expr] then
    Simple-command
fi
or
if[expr] then
  commandlist-1
else
  commandlist-2
if
```

The expression expr will be evaluated and according to its value, the commandlist-1 or the commandlist-2 will be executed.

```
Input:   $sh 1b.sh filename1 filename2

Output:
        Permissions of both files are same
        rwxr-xr-x
        rwxr-xr-x
        You have to change the file permission: $chmod 777 filename1
        Permissions are different
        rwxrwxrwx $+1
        rwxr_xr-x $+2
```

b) Write a shell script which accepts valid log in names as arguments and prints their corresponding home directories, if no arguments are specified, print a suitable error message.

```
clear

y=$#
i=1
if [ $y -eq 0 ]
then
  echo "arguments are not entered"
else
  while [ $i -le $y ]
do
 loginname=$1
 grep $loginname /etc/passwd > s
if [ $? -eq 0 ]
then
  echo "loginname:$loginname"
  echo "home directory"
cut -d ":" -f 6 s
fi
shift
i=`expr $i + 1`

done
fi
```

Description:
Grep: This command is used to search, select and print specified records or lines from an input file.
Grep [options] pattern [filename1][filename2]…
…….
Input: login name: MCA

Output: home directory
/home/MCA

……

3 a) Create a script file called file-properties that reads a file name entered and outputs it properties.

```
 echo "Enter a file name"
   read file
   if [ -f $file ]
 then
    set -- `ls -l $file`
 echo "file permission: $1"
 echo "Number of links:$2"
 echo "User name:$3"
 echo "Group name: $4"
 echo "Filesize : $5 bytes"
 echo "Date of modification:$6"
 echo "time of modification:$7"
 echo "Name of file:$8"
 else
    echo "file does not exist"
 fi
```

or

```
    #Check the permissions
    echo "enter the file name1"
    read f1
    ls -l $f1
```

……..
Input: Enter the file name
       biet
Output: -rwxrwxrwx   1 root root | mar | 11: 50 biet
……

b) Write shell script to implement terminal locking (similar to the lock command). It should prompt the user for a password. After accepting the password entered by the user, it must prompt again for the the matching password as confirmation and if match occurs, it must lock lock the keyword until a matching password is entered again by the user, Note that the script must be written to disregard BREAK, contro-D. No time limit need be implemented for the lock duration.

```
clear
echo "Enter the passwd for terminal locking"
stty -echo
read pass1
stty echo
echo "Enter passwd for confirmation"
stty -echo
read pass2
stty echo
val=1
while [ $val -eq 1 ]
do
if [ $pass2 = $pass1 ]
then
   echo "password match"
val=0
else
    echo "invalid password"
    echo "Enter password for confirmation"
    stty -echo
    read pass2
    stty echo
fi
    done
if [ $pass1 = $pass2 ]
   then
        echo "Terminal is locked"
   echo "Enter password to unlock terminal"
   stty -echo
   read pass3
   val=1
   while [ $val -eq 1 ]
   do
      while [ -z "$pass3" ]
      do
      sleep 1
      read pass3

done
```

```
     if [ $pass3 = $pass2 ]
     then
      val=0
else
clear
echo "invalid password"
echo "enter passwd for unlocking"
stty -echo
read pass3
fi
done
stty echo
fi

stty echo

echo "terminal unlocked"
```

……..

```
Input: Enter the password for terminal locking
       123
       Enter the password for confirmation
       123
       Enter password to unlock terminal
       123
 output: Terminal is unlocked
```

……..

4a. Write a shell script that accept one or more filenames as argument and convert all of them to uppercase, provided they exist in current directory.

```
y=$#
if [ $y -le 0 ]
then
     echo "argument is not entered"
else
     for file in $*
     do
echo "$file"
 n=`echo -n "$file" | tr "[a-z]"  "[A-Z]"`
mv "$file" "$n"
echo "$n"
done
fi
```

……..

```
Input: $sh 4b.sh Enter directory name

Output: All files change to uppercase
```

……..

b. Write a shell script that displays all the links to a file specified as the first argument to the script. The second argument, which is optional, can be used to specify in which the search is to begin. If this second argument is not present, the search is to begin in current working directory. In either case, the starting directory as well as all its subdirectories at all levels must be searched. The script need not include any error checking.

```
file=$1
if [ $# -eq 1 ]
then
  dirx="."
else
   dirx="$2"
 fi
set -- `ls -l $file`
lcnt=$2
if [ $lcnt -eq 1 ]
then
     echo "No other links"
     exit 0
 else
    set -- `ls -i $file`
   inode=$1
   find "$dirx" -xdev -inum $inode -print
fi
```

……….

```
Input: $sh sa.sh T1        Note: You have create a link $ln T1 T2

Output: ./y
        ./g
```

……….

5a. Write a shell script that accepts as filename as argument and display its creation time if file exist and if it does not send output error message.

```
if [ $# -eq 0 ]
then
   echo "display does not exit"
   else
      ls -l $1 > t1
x=`cut -c 42- t1`
echo $x
fi
```

……….

```
 input: $sh 5b.sh t1
 Ouput: 15:29 t1
```

……..

b. Write a shell script to display the calendar for current month with current date replaced by * or ** depending on whether the date has one digit or two digits.

```
   # program to display the current month and date

    set `date`
if [ $3 -le 9 ]
then
   n=`cal | tail -n +3 | grep -n "$3" | cut -d ":" -f1 | head -n1`
n=`expr $n + 2`
    cal|sed "$n s|$3|*|"
else
    cal|sed "s|$3|**|"
fi
```

……..

```
     Input: $sh 6a.sh

     Output:
         Mon Mar 17 09:39:20 IST 2008
         March 2008
         Su Mo Tu We Th Fr Sa
                           1
         2  3  4  5  6  7  8
         9 10 11 12 13 14 15
        16 ** 18 19 20 21 22
        23 24 25 26 27 28 29
        30 31
```

……

6a. Write a shell script to find a file/s that matches a pattern given as command line argument in the home directory, display the contents of the file and copy the file into the directory ~/mydir

```
if [ $# -eq 0 ]
  then
echo "No arguments"
exit
fi
for i in $*
 do
echo grep –riew $* /home/mca
ls $*
cat $*
 cp –f $* /home/mca/mydir
done
```

……..

Input: $ sh 6a.sh *.sh

Output: Copy the *.sh file to /home/mca/mydir
Display content *.sh files.

---

b. Write a shell script to list all the files in a directory whose filename is at least 10 characters. (use expr command to check the length)

```
echo "enter the string"
read str
le=`expr length $str`
  if [ $le -le 10 ]
  then
echo "String is less than or equal 10 characters"
else
 echo $str
 fi
```

……

```
Input: Enter the string
       biet
Output: String is less than 10 characters
```

……..

```
Clear
If [ $# -eq 0 ]
then
echo “enter directory name as argument”
else
c=’ls –l $* | cut –d “  “ –f  9’
echo “filename are $c”
for I in $c
do
len=`expr “$i” : ‘.*’`
if [ $len –ge 10 ]
then
echo “$i having $len”
fi
done
fi
```

7a. Write a shell script that gets executed displays the message either “Good Morning” or “Good Afternoon” or “Good Evening” depending upon time at which the user logs in.

```
hournow=`date | cut -c 12-13`
echo $hournow
user=`echo $LOGNAME | cut -d "/" -f 2`
echo $LOGNAME
case $hournow in
[0-1][0-1]|0[2-9])echo "good morning Mr/Ms: $LOGNAME";;
             1[2-9])echo "good after noon Mr/Ms: $LOGNAME";;
             1[6-9])echo "good Evening Mr/Ms: $LOGNAME";;
                *)echo "good Night Mr/Ms: $LOGNAME";;
esac
```

……

```
Input: $sh 9a.sh
Output: good after noon
```

……..

b. Write a shell script that accept a list of filenames as its argument, count and report occurrence of each word that is present in the first argument file on other argument files.

```
if [ $# -lt 2 ]
then
    echo "usage:wdcnt wordfile filename1 filename2......."
exit
   fi
for word in `cat $1`
do
for file in $*
do
if [ "$file" != "$1" ]
then
    echo "the word frequency of --$word--in file $file is:
`grep -iow "$word" $file | wc -w`"
fi
done
done
```

```
Input: $cat a
        Unix is an OS by the researchers for the researchers
        Unix is an OS of the present computing industry
       $cat b
        Os is an ss that acts as an interface between the
        human and the computer
       $sh 9b.sh  Wordfile a b          (Wordfile contains Unix)

     Ouput: The word frequency of Unix in the a is 2
            The word frequency of Unix in  b is 0
            The word frequency of OS in a is 2
            The word frequency of Os in b is 1
```

8a. Write a shell script that determine the period for which a specified user is working on system and display appropriate message.

```
echo "enter the login name of a user"
read name
userinfo=`who | grep -w "$name" | grep "pts"`
echo $userinfo
if [ $? -ne 0 ]
then
    echo "$name is not logged-in yet"
exit
fi
  hrs=`echo "$userinfo" | cut -c 34-35`
  echo "login time " $hrs
  min=`echo "$userinfo" | cut -c 37-38`
  echo "login Min" $min
  hrnow=`date | cut -c 12-13`
  echo "current hrs" $hrnow
  minnow=`date | cut -c 15-16`
  echo "cuurent Min" $minnow
if [ $minnow -lt $min ]
then
  minnow=`expr $minnow + 60`
```

```
  hrnow=`expr $hrnow + 1`
fi
  hour=`expr $hrnow - $hrs`
  minutes=`expr $minnow - $min`
  echo "Mr/Ms:$name is working since $hour hrs-$minutes minutes"
```

……..

```
Input:  Enter the login name
        MCA
Output: MCA is working since 1 hr – 13 minutes
```

……

b Write a shell script that reports the logging in of a specified user within one minute after he/she log in. The script automatically terminate if specified user does not log in during a specified period of time.

```
echo -n "enter the login name of the user".
read lname
period=0
echo -n "enter the unit of time(min):"
read min
until who | grep -w "$lname"> /dev/null
do
sleep 60
period=`expr $period + 1`
if [ $period -gt $min ]
then
     echo "$lname has not logged in since $min minutes."
exit
fi
done
echo "$lname has now logged in."
```

……

```
Input:  MCA
Output: MCA has now logged in
Input:  test1
Output: tetst1 has not logged in 1 minutes
```

…….

9a. Write a shell script that accept the file name, starting and ending line number as an argument and display all the lines between the given line number.

```
if [ $# -ne 3 ]
then
    echo "invalid number of arguments"
exit
  fi
      c=`cat $1 | wc -l`
     if [ $2 -le 0 -o $3 -le 0 -o $2 -gt $3 -o $3 -gt $c ]
     then
          echo "invalid input"
exit
  fi
      sed -n "$2,$3 p" $1
```

……..

```
Input: $cat proverb.txt
       A friend in need is a friend indeed
       gold
```

```
       biet
       Industry
       mca
ouput: $sh 11b.sh proverb.txt 2 4
       gold
       biet
       industry
```

.....

b. Write a shell script that folds long lines into 40 columns. Thus any line that exceeds 40 characters must be broken after 40th, a "\" is to be appended as the indication of folding and the processing is to be continued with the residue. The input is to be supplied through a text file created by the user.

```
echo "Enter the filename:\c"
read fn
for ln in `cat $fn`
do
  lgth=`echo $ln|wc -c`
  lgth=`expr $lgth - 1`
  s=1;e=40
  if [ $lgth -gt 40 ]
then
      while [ $lgth -gt 40 ]
      do
        echo "`echo $ln|cut -c $s-$e`\\"
        s=`expr $e + 1`
        e=`expr $e + 40`
        lgth=`expr $lgth - 40`
      done
      echo $ln|cut -c $s-
else
      echo $ln
fi
  done
echo "File folded"
```

.....

```
Input: $sh 12a.sh t.txt

Output:bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb/
       bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb/
       bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb/
       bbbbbbbbbbbbbbbbb
```

.....

10a Write an awk script that accepts date argument in the form of dd-mm-yy and displays it in the form if month, day and year. The script should check the validity of the argument and in the case of error, display a suitable message.

```
{  split ( $0, arr, "-")
  if ((arr[1] < 1) || (arr[1] >31) || (arr[2] < 1) || (arr[2] > 12)
{
     print "invalid date"
     exit 0
  }
    else {
     print arr[2]
      if (arr[2] == 1)
        print "Jan"
      if (arr[2] == 2)
        print "Feb"
      if (arr[2] == 3)
        print "March"
      if (arr[2] == 4)
        print "April"
      if (arr[2] == 5)
        print "May"
      if (arr[2] == 6)
        print "jun"
      if (arr[2] == 7)
        print "jul"
      if (arr[2] == 8)
       print "Aug"
      if (arr[2] == 9)
       print "sep"
      if (arr[2] == 10)
         print "oct"
      if (arr[2] == 11)
        print "Nov"
      if (arr[2] == 12)
        print "Dec"
print arr[3]
}
}
```

……….

```
Input: $awk -f 12b.awk
       03-15-2008
Output:
       March
       15
       2008
```

……

b. Write an awk script to delete duplicated line from a text file. The order of the original lines must remain unchanged.

```
BEGIN {
        print "Removing Duplicated lines"
}
  {
    line [++no] = $0
}
  END {
     for (i=1; i<=no; i++)
     {
    flag=1
    for (j=1; j<i; j++)
    if (line[i] == line[j])
    flag=0
    if (flag==1)
    print line[i] >> "out13a.txt"
 }
}
```

……..

```
Input: $cat t1.txt
        abcd
        abcd
        xyz
         (press ctrl d)
 Output: $cat 13a.txt
          xyz
```

…….

11a. Write an awk script to find out total number of books sold in each discipline as well as total book sold using associate array down table as given below.

Electrical	34
Mechanical	67
Electrical	80
Computer Science	43
Mechanical	65
Civil	98
Computer Science	64

```
BEGIN {print"Total number of books sold in each category"}
      {books [ $1 ]+=$2}
END { for(item in books)
    { printf("\t%-17s %ls %-5d\n", item, "=", books[item])
        total+=books[item]
    }
   printf("%-17s %ls %-5d\n", "Total books sold", "=", total)
}
```

…….

```
input: $awk -f 13b.awk
        Electrical  34
        Mechanical  67
        Electrical  80
        Computer Science 43
        Mechanical  65
        Civil       198
        Computer Science 64
```

```
Ouput: Electrical 114
       Mechanical 132
       Computer Science 107
       Civil 198
       Total: 551
```

……

b. Write an awk script to compute gross salary of an employee accordingly to rule given below.

If basic salary is < 10000 then HRA=15% of basic & DA=45% of basic
If basic salary is >=10000 then HRA=20% of basic & DA=50% of basic.

```
BEGIN { FS=":"
  print "\n\t\tsalary statement of employees for the month of feb"
        print "sl.no","\t","name","\t\t","designation","\t","BASIC",
              "\t","DA","\t","HRA","\t","GROSS"
     print}
   {slno++; basic_tot+=$5;
     if ( $5 >= 10000 )
        {da=0.45*$5; da_tot+=da;
           hra=0.15*$5;hra_tot+=hra;}

else { da=0.50*$5;da_tot+=da;
           hra=0.20*$5;hra_tot+=hra;}
          sal_tot+=$5 + da + hra
 printf "%2d\t%-15s %12-s %8d %8.2f %8.2f
%8.2f\n",slno,$2,$3,$5,da,hra,$5+da+hra}
     END {print "\n\ttotal basic paid is : rs " basic_tot
          print "\ttotal da paid is : rs " da_tot
          print "\ttotal hra paid is : rs " hra_tot
          print "total salary paid is : rs " sal_tot}
```

…….

```
         Input: $cat data.txt
                mc901:guru:md:3/2/2008:52000
                mc902:girish:ana:4/2/2008:30000
               $awk –f 14.awk data.txt
         Output:
salary statement of employees for the month of Februry
sl.No   Name          Designation    BASIC   DA      HRA     GROSS
 1     guru          md              52000 23400.00  7800.00 83200.00
 2     girish        ana             30000 13500.00  4500.00 48000.00

      Total basic paid is: Rs
      Total da paid is:Rs.
      Total hra paid is:Rs.
      Total salary paid is:Rs.
```

……..

Note: In the examination *each* student picks one question from a lot of *all the* 11 questions. *Question A & B Not to be included for examination*

D. Model Viva Voce questions

Unix	Unix is an Operating System. All Computers have operating systems. An operating system is a software that acts as an interface between the user and the computer hardware. An operating system acts as a resources manager. Here resources mean hardware resources like the processor, the main memory, the hard disk, I/O devices and other peripherals. In addition to being a multi user operating system, Unix gives its users, the feeling of working on an independent computer system.
Salient feature of Unix	1. **Unix is a multi-tasking** operating system – has the ability to support concurrent execution of two or more active processes. Here it may me noted that an instance of a program in execution is known as a process. 2. **Unix is a multi-user** operating system – has the ability to support more than one user to login into the system simultaneously and execute programs. 3. **Unix Operating system is highly portable**. Compared to other OS Unix on to different hardware platforms with minimal or no modifications at all.
Unix Components	1. The kernel 2. The shell 3. The file system ***The kernel*** is the heart of any Unix Operating system. This kernel is relatively a small piece of code that is embedded on the hardware. ***The shell*** is a program that sits on the kernel and acts as an agent or interface between the users and the kernel and hence the hardware.
Types of Shells	1.The Bourne Shell(Sh) 2. The C Shell(csh) 3. The Korn shell (ksh) 4. The Bourne-Again shell (bash)
The Shell prompts	$ (dollar) : Bourne and Korn Shells %(percent) : C shells #(hash) : Any shell as root
echo Command	The echo command is used to display messages. It is quite useful in developing interactive shell programs.
Aliases	Giving alternate names to commands
Unix files	A File is a sequence of bits, bytes or lines that is stored on a storage device like a disk. 1. Regular files 2. Directory files 3. Device files or special files
Hidden files	Dot files : Ex: .profile, .exec and other files.
File system: Dot(.) and Dot(..)	These are two special purpose file names that exist in every file system. The file name dot (.) refers to the directory that contains it and the file name dot-dot (..) refers to the parent of its current directory.
cat command	Create files and Display the contents of a file.
ls command	Listing of files : This command is used to list all the files in a current directory.
chmod command	Changing file permissions
chown Command	Changing the owner of a file
chgrep Command	Changing the group of a file
Redirection < > >> | 0 1 2	It is possible to change the source from where the input is taken by a program as well as destination to where the output is sent by a program. This mechanism of changing the input source and /or output destination is called redirection. < input redirection > output redirection >> output redirection with appending |(pipe) Connecting the output of one command as input to another command The file descriptors 0 1 and 2 are implicitly prefixed to the redirection operators. **0** is input descriptor **1** is output descriptor and **2 i**s standard error file.
Filter	A program or a command that reads its input from the standard input, processes it in some way, and writes its output to the standard output is called a filter. Many of the Unix Commands like cat, grep, tee, sort, more, head, tail, cut, paste and others are some examples of filters.
tee command	This is a mechanism that sends a copy of its input to one or more files as well as to the standard output.
/dev/tty and dev/null	/dev/tty : terminal file /dev/null: Trash files
cut command	Splitting files vertically –Using this command, required field (s) or column(s) can be extracted from a file.
Vi Editor	vi-stands for visual editor. It is full-screen editor. The three modes of the vi editor are the Command mode , the input mode and the ex mode.
.exrc file	This is an automatic initialization file, which will be present in the home directory. This is an optional file. This file contains a series of set commands with proper options as well as some other ex mode commands. **$cat > .exec** **set number**

grep Commnad	This command is used to search, select and print specified records or lines from an input file. **grep** is an acronym for globally search a regular expression and print it. **The syntax: grep [options] pattern [filename1] [filename2]…options –I, -v,-l, -n, -e. egrep : extended grep and fgrep: fixed grep.**
sed command	**Sed** is an acronym for the stream editor. It is an extremely powerful editor by using which, one can perform (affect) quick and easy changes to a file without entering into an editor like vi or emacs and others. General format: **$sed options 'address_actionlist' filelist**
Process	A **process** is an abstract concept of using which , one can explain, understand and control the execution of a program in an operating system. In its simplest form a process is defined as a program in execution.
Foreground and background process. Daemons	All the user processes, which are created by users with the shell, act upon the directions of the users and are normally attached to the terminal are called interactive processes. These types of processes are also called **foreground processes.** Certain processes can be made to run independent of terminals. Such processes that run without any attachment to a terminal are called non-interactive processes. These types of processes are also called **background processes.** All processes that keep running always without holding up any terminals and keep waiting for certain instructions either from the system or users and then immediately get into action are called **daemons.**
System variables and Local variables	System variables: are set either during the boot sequence or after logging in. examples PATH , HOME, MAIL, SHELL and TERM Variables. Local variables are user-defined variables.
$0	**$0** is a special shell variable that holds the parameter number 0 , the program name.
$# $* $@	**$#** : The variable $# holds a count of the total number of parameters, that is arguments. **$*** : Variable holds the list of all the arguments **$@:** Variable holds the list of arguments present in the command line
"$*" and "$@"	**"$*" :** the command treats all the command line arguments as a single argument. **"$@" :** the command treats the command line arguments as individual arguments.
test command	This is a built-in shell command that evaluates the expression given to it as an argument and returns true if the evaluation of the expression returns a zero or false If the evaluation returns non-zero. -eq : equal to -ne: not equal to -gt : greater than –ge: greater than or equal to -lt: less than –le : less than or equal to. -e file : true if file exists –r file: true if file exists and is readable. –h file : True if file exists and is a link file.
Loop control structures	While , until, for
awk	**awk** is a filter program that was originally developed in 1977 by Aho Weinberger and Kernighan as a pattern-scanning language. The name awk is derived from the first letters of its developers surname . It is a programming language with C-like control structures, functions and variables. The general format of an awk command line is : **$ awk options 'program' filelist** **FS :** Input field separator (default: blank and tab) **OFS:** Output field separator (default: blank and tab) **RS** : input record separator (default: new line) **NR:** Number of current record NF: Number of fields in input record ARGC: Number of command line arguments ARGV: Command line arguments array. $0: Entire current line
Superuser	A **superuser** is a user with unrestricted access to all files and commands, The username of the superuser's account is root. Many administrative tasks and their associated commands require superuser status.
df and du commands	The **df(disk free)** command is used to find the amount of disk space available on a file system. The **du(disk usage)** command is used to find out how much disk space has been used by each sub-directory as well as each file under the current directory.
find command	**find i**s the Unix's file search command using which, one can search a required file in any required directory structure or directory structures. This command is one of the least used and most powerful commands in the Unix environments. **#find path_list selection_criterion action**
ln command Hard links Symbolic links	**ln** command : file links **Hard links**: In Unix more than one user can use the same file with his or her own filename. $ln trial test **Symbolic links** are files that hold the pathname of the original file. Since symbolic links are also files the inode numbers of these linked files will be different. One of the common uses of symbolic links is on the web.