LAB MANUAL

Course : CP1644 FOSS Lab

Class : BCA

Semester : 6

Prepared By : Syama M Nair

COLLEGE OF APPLIED SCIENCE

PERISSERY

TABLE OF CONTENTS

FAMILIARIZATION OF LINUX COMMANDS

* date Command

Display the current time in the given FORMAT, or set the system date.

Usage: date [OPTION]... [+FORMAT]

or: date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

Options

-d, --date=STRING display time described by STRING, not 'now'

-f, --file=DATEFILE like --date once for each line of DATEFILE

-r, --reference=FILE display the last modification time of FILE

-R, --rfc-2822 output date and time in RFC 2822 format.

-u, --utc, --universal print or set Coordinated Universal Time.

* mkdir Command

Create the DIRECTORY(ies), if they do not already exist.

Usage: mkdir [OPTION]... DIRECTORY...

Options

-m, --mode=MODE set file mode , not a=rwx – umask

-p, --parents no error if existing, make parent directories as needed

-v, --verbose print a message for each created directory

--help display this help and exit

--version output version information and exit

* rmdir Command

Remove the DIRECTORY(ies), if they are empty.

Usage: rmdir [OPTION]... DIRECTORY...

-p, --parents remove DIRECTORY and its ancestors;

-v, --verbose output a diagnostic for every directory processed

--help display this help and exit

--version output version information and exit

* bc Command

usage: bc [options] [file ...]

-h --help print this usage and exit

-i --interactive force interactive mode

-l --mathlib use the predefined math routines

-q --quiet don't print initial banner

-s --standard non-standard bc constructs are errors

-w --warn warn about non-standard bc constructs

-v --version print version information and exit

* passwd Command

Change user password.

Usage: passwd [options] [LOGIN]

Options:

-a, --all report password status on all accounts

-d, --delete delete the password for the named account

-e, --expire force expire the password for the named account

-k, --keep-tokens change password only if expired

-q, --quiet quiet mode

-S, --status 12report password status on the named account

-u, --unlock unlock the password of the named account

* who Command

Print information about users who are currently logged in.

Usage: who [OPTION]... [FILE | ARG1 ARG2]

Options

-a, --all same as -b -d --login -p -r -t -T -u

-b, --boot time of last system boot

-d, --dead print dead processes

-l, --login print system login processes

-m only hostname and user associated with stdin

-q, --count all login names and number of users logged on

-r, --runlevel print current runlevel

-u, --users list users logged in

* tty Command

Print the file name of the terminal connected to standard input.

Usage: tty [OPTION]...

Options

-s, --silent, --quiet print nothing, only return an exit status

--help display this help and exit

--version output version information and exit

* ls Command

List information about the FILEs (the current directory by default).

Usage: ls [OPTION]... [FILE]...

Options

-a, --all do not ignore entries starting with .

-A, --almost-all do not list implied . and ..

--author with -l, print the author of each file

-b, --escape print C-style escapes for nongraphic characters

-C list entries by columns

-d, --directory list directory entries instead of contents,

and do not dereference symbolic links

* cat Command

Concatenate FILE(s), or standard input, to standard output

Usage: cat [OPTION]... [FILE]...

Options

-A, --show-all equivalent to -vET

-e equivalent to -vE

-E, --show-ends display $ at end of each line

-n, --number number all output lines

-t equivalent to -vT

* cp Command

Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

Usage: cp [OPTION]... [-T] SOURCE DEST

or: cp [OPTION]... SOURCE... DIRECTORY

or: cp [OPTION]... -t DIRECTORY SOURCE...

Options

-a, --archive same as -dR --preserve=all

-b like --backup but does not accept an argument

-d same as --no-dereference --preserve=links

-i, --interactive prompt before overwrite

* rm Command

Remove (unlink) the FILE(s).

Usage: rm [OPTION]... FILE...

Options

-f, --force ignore nonexistent files and arguments, never prompt

-i prompt before every removal

-I prompt once before removing more than three files, or
when removing recursively.

-d, --dir remove empty directories

* mv Command

Usage: mv [OPTION]... [-T] SOURCE DEST

or: mv [OPTION]... SOURCE... DIRECTORY

or: mv [OPTION]... -t DIRECTORY SOURCE...

Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

Options

-b like --backup but does not accept an argument

-f, --force do not prompt before overwriting

-i, --interactive prompt before overwrite

-n, --no-clobber do not overwrite an existing file

* wc Command

Usage: wc [OPTION]... [FILE]...

or: wc [OPTION]... --files0-from=F

Print newline, word, and byte counts for each FILE, and a total line if more than one FILE is specified. With no FILE, or when FILE is -, read standard input. A word is a non-zero-length sequence of characters delimited by white space.

Options

-c, --bytes print the byte counts

-m, --chars print the character counts

-l, --lines print the newline counts

-L, --max-line-length print the length of the longest line

-w, --words print the word counts

* cmp Command

Usage: cmp [OPTION]... FILE1 [FILE2 [SKIP1 [SKIP2]]]

Compare two files byte by byte.

Options

-b, --print-bytes print differing bytes

-i, --ignore-initial=SKIP skip first SKIP bytes of both inputs

-l, --verbose output byte numbers and differing byte values

-n, --bytes=LIMIT compare at most LIMIT bytes

-s, --quiet, --silent suppress all normal output

* comm Command

Usage: comm [OPTION]... FILE1 FILE2

Compare sorted files FILE1 and FILE2 line by line.

Options

--check-order check that the input is correctly sorted, even if all input lines are pairable

--nocheck-order do not check that the input is correctly sorted

--output-delimiter=STR separate columns with STR

* chmod Command

Usage: chmod [OPTION]... MODE[,MODE]... FILE...

or: chmod [OPTION]... OCTAL-MODE FILE...

or: chmod [OPTION]... --reference=RFILE FILE...

Change the mode of each FILE to MODE.

Options

-c, --changes like verbose but report only when a change is made

-f, --silent, --quiet suppress most error messages

-R, --recursive change files and directories recursively

* head Command

Usage: head [OPTION]... [FILE]...

Print the first 10 lines of each FILE to standard output.

* tail Command

Usage: tail [OPTION]... [FILE]...

Print the last 10 lines of each FILE to standard output.

Options

-c, --bytes=K output the last K bytes;

-f output appended data as the file grows;

* sort Commands

Usage: sort [OPTION]... [FILE]...

or: sort [OPTION]... --files0-from=F

Write sorted concatenation of all FILE(s) to standard output.

Options

-b, --ignore-leading-blanks ignore leading blanks

-d, --dictionary-order consider only blanks and alphanumeric characters

-f, --ignore-case fold lower case to upper case characters

-g, --general-numeric-sort compare according to general numerical value

-i, --ignore-nonprinting consider only printable characters

INTRODUCTION TO VI-EDITOR.

The default editor that comes with the UNIX operating system is called vi (**vi**sual editor)

The UNIX vi editor is a full screen editor and has two modes of operation:

1. ***Command mode*** commands which cause action to be taken on the file, and
2. ***Insert mode*** in which entered text is inserted into the file.

In the command mode, every character typed is a command that does something to the text file being edited; a character typed in the command mode may even cause the vi editor to enter the insert mode. In the insert mode, every character typed is added to the text in the file; pressing the <Esc> (*Escape*) key turns off the Insert mode.
NOTE: Both UNIX and vi are **case-sensitive**. Be sure not to use a capital letter in place of a lowercase letter; the results will not be what you expect.

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

*	vi filename	*edit* filename *starting at line 1*
	vi -r filename	*recover* filename *that was being edited when system crashed*

To Exit vi

Usually the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to bottom of screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

*	:x<Return>	*quit* vi*, writing out modified file to file named in original invocation*
	:wq<Return>	*quit* vi*, writing out modified file to file named in original invocation*
	:q<Return>	*quit (or exit)* vi
*	:q!<Return>	*quit* vi *even though latest changes have not been saved for this* vi *call*

Moving the Cursor

Unlike many of the PC and Macintosh editors, **the mouse does not move the cursor** within the vi editor screen (or window). You must use the key commands listed below. On some UNIX platforms, the arrow keys may be used as well; however, since vi was designed with the Qwerty keyboard (containing no arrow keys) in mind, the arrow keys sometimes produce strange effects in viand should be avoided.

In the table below, the symbol ^ before a letter means that the <Ctrl> key should be held down while the letter key is pressed.

*	j ***or*** <Return> **[*or* down-arrow]**	*move cursor down one line*
*	k **[*or* up-arrow]**	*move cursor up one line*
*	h ***or*** <Backspace> **[*or* left-arrow]**	*move cursor left one character*
*	l ***or*** <Space> **[*or* right-arrow]**	*move cursor right one character*
*	0 **(zero)**	*move cursor to start of current line (the one with the cursor)*
*	$	*move cursor to end of current line*
	W	*move cursor to beginning of next word*
	B	*move cursor back to beginning of preceding word*
	:0<Return> or 1G	*move cursor to first line in file*
	:n<Return> or nG	*move cursor to line* n
	:$<Return> or G	*move cursor to last line in file*

Screen Manipulation

The following commands allow the vi editor screen (or window) to move up or down several lines and to be refreshed.

^f	*move forward one screen*
^b	*move backward one screen*
^d	*move down (forward) one half screen*
^u	*move up (back) one half screen*
^l	*redraws the screen*
^r	*redraws the screen, removing deleted lines*

Adding, Changing, and Deleting Text

Unlike PC editors, you cannot replace or delete text by highlighting it with the mouse. Instead use the commands in the following tables.

Perhaps the most important command is the one that allows you to back up and *undo* your last action. Unfortunately, this command acts like a toggle, undoing and redoing your most recent action. You cannot go back more than one step.

u	*UNDO WHATEVER YOU JUST DID; a simple toggle*

The main purpose of an editor is to create, add, or modify text for a file.

Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the <Esc> key must be pressed to terminate the entry of text and to put the vieditor back into command mode.

*	i	*insert text before cursor, until* <Esc> *hit*
	I	*insert text at beginning of current line, until* <Esc> *hit*
*	a	*append text after cursor, until* <Esc> *hit*
	A	*append text to end of current line, until* <Esc> *hit*
*	o	*open and put text in a new line below current line, until* <Esc> *hit*
*	O	*open and put text in a new line above current line, until* <Esc> *hit*

Changing Text

The following commands allow you to modify text.

*	R	*replace single character under cursor (no* <Esc> *needed)*
	R	*replace characters, starting with current cursor position, until* <Esc> *hit*
	Cw	*change the current word with new text, starting with the character under cursor, until* <Esc> *hit*
	cNw	*change* N *words beginning with character under cursor, until* <Esc> *hit; e.g.,* c5w *changes 5 words*
	C	*change (replace) the characters in the current line, until* <Esc> *hit*
	Cc	*change (replace) the entire current line, stopping when* <Esc> *is hit*
	Ncc ***or*** cNc	*change (replace) the next N lines, starting with the current line, stopping when* <Esc> *is hit*

Deleting Text

The following commands allow you to delete text.

*	X	*delete single character under cursor*
	Nx	*delete N characters, starting with character under cursor*
	Dw	*delete the single word beginning with character under cursor*
	dNw	*delete* N *words beginning with character under cursor; e.g.,* d5w *deletes 5 words*
	D	*delete the remainder of the line, starting with current cursor position*
	Dd	*delete entire current line*
	Ndd ***or*** dNd	*delete* N *lines, beginning with the current line; e.g.,* 5dd *deletes 5 lines*

Cutting and Pasting Text

The following commands allow you to copy and paste text.

	Yy	*copy (yank, cut) the current line into the buffer*
	Nyy ***or*** yNy	*copy (yank, cut) the next N lines, including the current line, into the buffer*
	P	*put (paste) the line(s) in the buffer into the text after the current line*

Other Commands

Searching Text

A common occurrence in text editing is to replace one word or phase by another. To locate instances of particular sets of characters (or strings), use the following commands.

	/string	*search forward for occurrence of* string *in text*
	?string	*search backward for occurrence of* string *in text*
	N	*move to next occurrence of search string*
	N	*move to next occurrence of search string in opposite direction*

Determining Line Numbers

Being able to determine the line number of the current line or the total number of lines in the file being edited is sometimes useful.

	:.=	*returns line number of current line at bottom of screen*
	:=	*returns the total number of lines at bottom of screen*
	^g	*provides the current line number, along with the total number of lines, in the file at the bottom of the screen*

Saving and Reading Files

These commands permit you to input and output files other than the named file with which you are currently working.

	:r filename<Return>	*read file named* filename *and insert after current line (the line with cursor)*
	:w<Return>	*write current contents to file named in original* vi *call*
	:w newfile<Return>	*write current contents to a new file named* newfile
	:12,35w smallfile<Return>	*write the contents of the lines numbered 12 through 35 to a new file named* smallfile
	:w! prevfile<Return>	*write current contents over a pre-existing file named* prevfile

PROGRAM 1:

AIM:- *Write a shell program to find the average of n numbers.*

PROGRAM:

```
echo " Enter the limit ";
read l;
echo " Enter the numbers";
for((i=0;i<l;i++))
        do
        read a[$i];
        done

sum= 0;
for((i=0;i<l;i++))
        do
        sum=$(($sum + ${a[$i]}));
        done

n=` expr $sum / $l `;
echo " Sum is "$sum;
echo "Average is "$n;
```

OUTPUT

```
Enter the limit
4
Enter the numbers
1
2
3
4
Sum is 10
Average is 2
```

PROGRAM 2:

AIM:- *Write a shell program to find the number is even or odd.*

PROGRAM:

```
echo "Enter a number";
read a;
n=` expr $a % 2 `;
if [ $n -eq 0 ]
then
echo "$a is an even number ";
else
echo "$a is an odd number";
fi
```

OUTPUT

```
Enter a number
3
3 is an odd number
Enter a number
6
6 is an even number
```

PROGRAM 3:

AIM:- *Write a shell program to print the Multiplication table.*

PROGRAM:

```
echo "Enter a number";
read a;
echo "Multiplication table of $a is:"
for (( i=1; i<=10; i++))
do
n=` expr $i \* $a `;
echo " $a * $i = $n ";
done
```

OUTPUT

```
Enter a number
4
Multiplication table of 4 is:
 4 * 1 = 4
 4 * 2 = 8
 4 * 3 = 12
 4 * 4 = 16
 4 * 5 = 20
 4 * 6 = 24
 4 * 7 = 28
 4 * 8 = 32
 4 * 9 = 36
 4 * 10 = 40
```

PROGRAM 4:

AIM:- *Write a shell program to find the largest of three numbers.*

PROGRAM:

```
echo "Enter three Integers:"
read a
read b
read c

if [ $a -gt $b -a $a -gt $c ]
then
   echo "$a is Greatest!"
elif [ $b -gt $c -a $b -gt $a ]
then
   echo "$b is Greatest!"
else
   echo "$c is Greatest!"
fi
```

OUTPUT

```
Enter three Integers:
5
4
3
5 is Greatest !
```

PROGRAM 5:

AIM:- *Write a shell program to find the sum of digits of a number.*

PROGRAM:

```
echo "Enter a Number:"
read n

temp=$n
sd=0
sum=0

while [ $n -gt 0 ]
   do
      sd=$(( $n % 10 ))
      n=$(( $n / 10 ))
      sum=$(( $sum + $sd ))
   done
echo "Sum is $sum"
```

OUTPUT

```
Enter a Number:
542
Sum is 11
```

PROGRAM 6:

AIM:- *Write a shell program to check whether the year is leap year or not.*

PROGRAM:

```
echo "Enter Year:"
read y
year=$y
y=$(( $y % 4 ))
if [ $y -eq 0 ]
then
   echo "$year is Leap Year!"
else
   echo "$year is not a Leap Year!"
fi
```

OUTPUT

```
Enter Year:
2016
2016 is Leap Year!
Enter Year:
2017
2017 is not a Leap Year!
```

PROGRAM 7:

AIM:- *Write a shell program to check whether the number is palindrome or not.*

PROGRAM:

```
echo "Enter the number"
read n
number=$n
reverse=0
while [ $n -gt 0 ]
do
a=`expr $n % 10 `
n=`expr $n / 10 `
reverse=`expr $reverse \* 10 + $a`
done
echo $reverse
if [ $number -eq $reverse ]
then
echo "Number is palindrome"
else
echo "Number is not palindrome"
fi
```

OUTPUT

Enter the number

232

232

Number is palindrome

Enter the number

321

123

Number is not palindrome

PROGRAM 8:

AIM:- *Write a shell program to check whether the number is prime or not.*

PROGRAM:

```
echo "Enter the number"
read a;
flag=0
for (( i=2; i<a; i++ ))
do
b=`expr $a % $i`
if [ $b -eq 0 ]
then
flag=1
fi
done
if [ $flag -eq 1 ]
then
echo "$a is not a prime number "
else
echo "$a is a prime number "
fi
```

OUTPUT

Enter the number

2

2 is a prime number

Enter the number

4

4 is not a prime number

PROGRAM 9:

AIM:- *Write a shell program to find the factorial of a number.*

PROGRAM:

```
echo "Enter a number"
read n
fact=1
for((i=$n;i>=1;i--))
do
fact=`expr $fact \* $i`
done
echo "The factorial of $n is $fact"
```

OUTPUT

```
Enter a number
3
The factorial of 3 is 6
```

PROGRAM 10:

AIM:- *Write a shell program to print the Fibonacci series up to N.*

PROGRAM:

```
echo "Enter the limit"
read Num
f1=0
f2=1
echo "The Fibonacci sequence for the number $Num is : "
for (( i=0;i<Num;i++ ))
do
    echo -n "$f1 "
    fn=$((f1+f2))
    f1=$f2
    f2=$fn
done
```

OUTPUT

```
Enter the limit
6
The Fibonacci sequence for the number 6 is :
0 1 1 2 3 5
```

PROGRAM 11:

AIM:- *Write a shell program to check the number is Armstrong or not.*

PROGRAM:

```
echo "enter the number"
read n
m=` expr $n `
s=0
while [ $n -gt 0 ]
do
x=` expr $n % 10 `
d=` expr $x \* $x \* $x `
s=` expr $s + $d `
n=` expr $n / 10 `
done
if [ $m -eq $s ]
then
echo "armstrong number"; 16
else
echo "not amstrong number";
fi
```

OUTPUT

Enter the number

153

Armstrong number

Enter the number

234

Not amstrong number

PROGRAM 12:

AIM:- *Write a shell program to copy the contents of two files.*

PROGRAM:

```
echo "Enter the file name"
read k
echo "Enter the destination file"
read n
if [ -f $k ]
then
cp $k $n
echo "File copied"
else
echo "Not Exist"
fi
```

OUTPUT

```
Enter the file name
upper.sh
Enter the destination file
r1.sh
File copied
```

PROGRAM 13:

AIM:- *Write a shell program to print system configurations like.*

1) Currently logged user and his logname
2) Your current shell
3) Your home directory
4) Your current path setting
5) Your current working directory
6) Show Currently logged number of users

PROGRAM:

```
nouser=`who | wc -l`

echo "Username: $USER (Login name: $LOGNAME)"

echo "Current shell: $SHELL"

echo "Home Directory:$HOME"

echo "Path: $PATH"

echo "Current directory: `pwd`"

echo "Currently Logged:$nouser user(s)"
```

OUTPUT

```
Username: acer (Login name: acer)

Current shell: /bin/bash

Home Directory:/home/acer

Path: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games

Current directory: /home/acer/fair

Currently Logged:2 user(s)
```

PROGRAM 14:

AIM:- *Write a shell program to print the pattern.*

PROGRAM:

```
makePyramid()
{
   n=$1;
 for((i=1;i<=n;i++)) Enter the user name
guest

guest is not logged in

Enter the user name

acer

acer is logged in
  do
     for((k=i;k<=n;k++))
     do
       echo -ne " ";
     done
    for((j=1;j<=2*i-1;j++))
     do
          echo -ne "*"
     done
         echo;
  done
}
makePyramid 5
```

OUTPUT

```
      *
     ***
   *****
  *******
*********
```

PROGRAM 15:

AIM:- *Write Script to see current date, time, username, and current directory.*

PROGRAM:

```
echo "Hello, $LOGNAME"
echo "Current date is `date`"
echo "User is `who i am`"
echo "Current direcotry `pwd`"
```

OUTPUT

```
Hello, acer
Current date is Mon Dec  4 11:41:32 IST 2017
User is acer     pts/0        2017-12-04 11:18 (:0)
Current direcotry /home/acer/fair
```

PROGRAM 16:

AIM:- *Write a shell program to check whether a given user is logged in or not.*

PROGRAM:

```
echo "Enter the user name"
read name
if who | grep $name > /dev/null
then
echo "$name is logged in"
else
echo "$name is not logged in"
fi
```

OUTPUT

```
Enter the user name
guest
guest is not logged in
Enter the user name
acer
acer is logged in
```

PROGRAM 17:

AIM:- *Write a shell program to perform arithmetic operations using function.*

PROGRAM:

```
echo "Enter two numbers"
read a
read b
addition()
{
c=$((a + b))
echo "Sum is"$c
}
subtraction()
{
c=$((a - b))
echo "Difference is"$c
}
multiplication()
{
c=$((a * b))
echo "Product is"$c
}
division()
{
c=$((a / b))
echo "Coifitient is"$c
}
echo “1.Addition 2.Subtraction 3.Multiplication 4.Division 5.Exit”
echo "Enter your option"
read op

case $op in

1) addition ;;

2) subtraction ;;

3) multiplication ;;

4) division ;;

5) exit
esac
```

OUTPUT

Enter two numbers

12

4

“1.Addition 2.Subtraction 3.Multiplication 4.Division 5.Exit”

Enter your option

1

Sum is16