DEPARTMENT OF COMPUTER ENGINEERING & APPLICATIONS

Institute of Engineering & Technology

Lab Manual

Subject Name & Code: UNIX Lab / ITE 381

Faculty: Jitesh Kumar Bhatia

Course: B. Tech (CSE) **Session: 2014-15**
Year: III **Semester: VI**

INDEX

EXPERIMENT NO: 1

Environment: Microsoft Windows/Linux

Objective: Implement all basic commands used in Linux OS

a. ls	**e. man**	**i. rmdir**
b. mkdir	**f. date**	**j. head**
c. cd	**g. cal**	**k. tail**
d. cat	**h. rm**	**l. pwd**

Procedure/Algorithm/Steps/Syntax:

The commands used in UNIX/LINUX are:

S. No.	Command	Example	Implementation
a.	**ls**	ls	Lists files in current directory
		ls -a	List all files in current directory
b.	**mkdir**	mkdir graphics	Make a directory called graphics
c.	**cd**	cd tempdir	Change directory to tempdir
		cd ..	Move back one directory
		cd ~gla/student	Move into gla's student directory
d.	**cat**	cat filename	Opens an existing file
e.	**man**	man ls	Online manual (help) about command
f.	**date**	date	Print out current date
g.	**cal <mo> <yr>**	cal 9 2000	Print calendar for September 2000
h.	**rm**	rm file1.bak	Remove or delete file
		rm *.tmp	Remove all file with .tmp extension
i.	**rmdir**	rmdir emptydir	Remove directory (must be empty)
j.	**head**	head <filename>	Displays first 10 lines of the file
k.	**tail**	tail <filename>	Displays last 10 lines of the file
l.	**pwd**	pwd	Prints current working directory

Code:

As according to the table given above

Sample Input:

As according to the table given above

Sample Output:

As according to the table given above

Post Experiment Question:

1. Is Linux GUI based or CUI based?
2. Is UNIX GUI based or CUI based?
3. What is the use of ls, cat, head and tail?
4. What are read write execute permissions on a directory?
5. What is process based commands?

EXPERIMENT NO: 2

Environment: Microsoft Windows/Linux

Objective: Implement all basic commands used in Linux OS

a. cp	**e. who**	**i. bc**
b. mv	**f. whoami**	**j. top**
c. sort	**g. ps**	**k. grep**
d. cut	**h. kill**	**l. chmod**

Procedure/Algorithm/Steps/Syntax:

The commands used in UNIX/LINUX are:

S. No.	Command	Example	Implementation
a.	**cp**	cp file1 file2	Copy file into directory
		cp file1 file1.bak	Make backup of file1
b.	**mv**	mv file1 file2	Move or rename files
c.	**sort**	sort file1	Sorts the contents of the file alphabetically
		sort –n file1	Sorts the contents of the file numerically
		sort –r file1	Sorts the contents of the file alphabetically in reverse order
d.	**cut**	cut –m -n filename	Displays only the columns from m to n of the file
e.	**who**	who	Lists who is logged on your machine
f.	**whoami**	whoami	Lists the username of the current user logged in.
g.	**ps <opt>**	ps aux	List all running processes by #ID
		ps aux | grep dhyatt	List process #ID's running by dhyatt
h.	**kill <opt> <ID>**	kill -9 8453	Kill process with ID #8453
i.	**bc**	bc	starts a calculator
j.	**top**	top	Various fields are displayed as per the current scenario
k.	**grep <str><files>**	grep "bad word" file1	Find which lines in a file contain a certain word
l.	**chmod <opt> <file>**	chmod 644 *.html	Change file permissions read only
		chmod 755 file.exe	Change file permissions to executable

Options with ps command

Various options for ps are:

Option	Description
-a	Displays all processes on a terminal, with the exception of group leaders.
-c	Displays scheduler data.
-d	Displays all processes with the exception of session leaders.
-e	Displays all processes.
-f	Displays a full listing.
-g*list*	Displays data for the *list* of group leader IDs.
-j	Displays the process group ID and session ID.
-l	Displays a long listing
-p*list*	Displays data for the *list* of process IDs.
-s*list*	Displays data for the *list* of session leader IDs.
-t*list*	Displays data for the *list* of terminals.
-u*list*	Displays data for the *list* of usernames.

Details of top command

On implementing top command, the following fields are displayed:

1. CODE -- Code Size (KB).
2. COMMAND -- Command Name or Command Line.
3. %CPU -- CPU Usage.
4. DATA -- Data + Stack Size (KB).
5. Flags -- Task Flags.
6. GID -- Group Id.
7. GROUP -- Group Name.
8. %MEM -- Memory Usage
9. NI -- Nice Value.
10. nDRT -- Dirty Pages Count.
11. nMaj -- Major Page Fault Count.
12. nMin -- Minor Page Fault count.
13. nTH -- Number of Threads.
14. PID -- Process Id.
15. PPID -- Parent Process Id.
16. RES -- Resident Memory Size (KB), etc.

Code:

As according to the table given above

Sample Input:

As according to the table given above

Sample Output:

As according to the table given above

Post Experiment Question:

1. Explain the syntax of ps with any 2 options.
2. What are the various options of sort command?
3. What are the various options of bc command?
4. What is the meaning of obase and ibase?
5. What is the significance of scale?
6. What is the significance of grep command?
7. What are the various options used in grep command?

EXPERIMENT NO: 3

Environment: Microsoft Windows/Linux

Objective: Implement all basic commands used in Linux OS

a. nohup	**e. pg**	**i. sed**
b. last	**f. history**	**j. emacs**
c. more	**g. touch**	**k. du**
d. less	**h. tr**	**l. diff**

Procedure/Algorithm/Steps/Syntax:
The commands used in UNIX/LINUX are:

S. No.	Command	Example	Implementation
a.	**nohup**	nohup find -size +100k > log.txt	Runs a command, making it immune to any HUP (hangup) signals while it is running. Runs the find command, instructing it to search for any file bigger then 100k. Find will continue to search regardless of the user's connection to the terminal, and log results to the file log.txt
b.	**last**	last	Finds out who has recently logged in and out on your server
c.	**more**	more file1	Look at file, one page at a time
d.	**less**	less file1	Less is a program similar to more, but which allows backward movement in the file as well as forward movement.
e.	**pg**	pg <filename>	pg displays a text file on a CRT one screenful at once. After each page, a prompt is displayed. The user may then either press the newline key to view the next page
f.	**history**	history	Lists commands you've done recently
g.	**touch**	touch <filename>	Creates a new file of zero bytes, if does not exist. If, however exists, changes the timestamp of it.
h.	**tr**	tr [a-z] [A-Z] <filename>	Translates in the file as per given translation. In the example, changes the lowercase characters to uppercase characters.

i.	**sed**	sed 's/unix/linux/' file.txt	Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.
j.	**emacs**	emacs filename	emacs is a screen editor.
k.	**du**	du <filename>	Lists the disk usage of the given file.
l.	**diff**	diff <file1> <file2>	Checks whether two files are same or different.

Code:
As according to the table given above

Sample Input:
As according to the table given above

Sample Output:
As according to the table given above

Post Experiment Question:

1. Explain the syntax of nohup.
2. How does diff command work?
3. What does history command give as output?
4. What is the difference between tr and sed?
5. Where does the output of the command given with nohup gets stored by default?

EXPERIMENT NO: 4

Environment: Microsoft Windows/Linux

Objective: Write and implement the basic vi editor commands.

Procedure/Algorithm/Steps/Syntax:
vi is a visual text editor.
vi is actually the command which launches the visual mode of ex. As ex gained popularity, he noticed that most users were exclusively using its visual mode, so he created a link to ex which immediately starts in visual mode and simply named it vi. The name stuck, and today vi is the most popular text editor among Linux users.

Code:
vi [option]

-l	Set up for editing LISP programs.
-R	Readonly mode; the readonly flag is set, preventing accidental overwriting of the file.
-r filename	Edit filename after an editor or system crash. (Recovers the version of filename that was in the buffer when the crash occurred.)
-S	This option is used in conjunction with the -t tag option to tell vithat the tags file may not be sorted and that, if the binary search (which relies on a sorted tags file) for tag fails to find it, the much slower linear search should also be done. Since the linear search is slow, users of large tags files should ensure that the tags files are sorted rather than use this flag. Creation of tags files normally produces sorted tags files. See ctags for more information on tags files.
-t tag	Edit the file containing the tag tag, and position the editor at its definition.
-v	Start up in display editing state using vi. You can achieve the same effect by typing the vi command itself.
-V	Verbose mode. When ex commands are read by means ofstandard input, the input will be echoed to standard error. This may be useful when processing ex commands within shell scripts.
-x	Encryption option; when used, vi simulates the X command of exand prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of the crypt command.
-wn	Set the default window size to n. This is useful when using the editor over a slow speed line.

-C	Encryption option; same as the -x option, except that visimulates the C command of ex. The C command is like the Xcommand of ex, except that all text read in is assumed to have been encrypted.
+command | -c command	Begin editing by executing the specified editor command (usually a search or positioning command).
filename	The file to be edited.

Sample Input:

vi myfile.txt

Sample Output:
Edits the file myfile.txt

Post Experiment Question:

1. Why is vi so popular?
2. What are the various modes of vi editor?

EXPERIMENT NO: 5

Environment: Microsoft Windows/Linux

Objective: Shell scripts that uses simple commands

a. Write a shell script to display current date in a particular format, number of users currently login and current month's calendar.

Procedure/Algorithm/Steps/Syntax:
The current date is displayed using the date command.
The number of users currently login is displayed using who|wc –l command.
The current month's calendar is displayed using cal command.

Code:
```
echo "Current date is"; date
echo "Number of users"; who|wc –l
echo "Monthly calendar"; cal
```

Save it as file1.sh

Sample Input:
```
chmod 754 file1.sh
./file1.sh
```

Sample Output:
Current date is <today's date>
Number of users 20
Monthly calendar
<Current month's calendar>

b. Write a shell script to display the process name and its pid.

Procedure/Algorithm/Steps/Syntax:
The process id (PID) is displayed using $$.
The process name is displayed using $0.

Code:
```
echo "Current process' name: $0"
echo "Current process' id: $$"
```

Save it as file1.sh

Sample Input:
```
chmod 754 file1.sh
./file1.sh
```

Sample Output:
Current process' name: ./file1.sh
Current process' id: 1245 (Or any other number)

Post Experiment Question:

1. What is a shell script? How do we run it?
2. What is the use of chmod command? Explain the options used in it.
3. Why does $0 display the current shell script name?
4. Is the answer same all the time for the same process? Why or why not?

EXPERIMENT NO: 6

Environment: Microsoft Windows/Linux

Objective: Decision based Shell scripts:

a. Write a shell script that finds whether an entered number is even or odd.

Procedure/Algorithm/Steps/Syntax:
The % operator is used to find the remainder.
The if [condition] then statement else statement is the syntax of if-else construct.
fi is used to end the if statement.

Code:

```
echo –n ”Enter a number”
read x
rem=$ (( $x % 2 ))
if [ $rem -eq 0 ]
then
echo “$x is even”
else
echo “$x is odd”
fi
```

Save it as file1.sh

Sample Input:
chmod 754 file1.sh
./file1.sh

Sample Output:
Enter a number
10
10 is even

b. Write a shell script to input the name of a file as command line argument and display whether it is a file, a directory or anything else.

Procedure/Algorithm/Steps/Syntax:
The –f option checks for a file and the –d option checks for a directory.
The first command line argument is stored in the variable $1, the second in $2, and so on.
fi is used to end the if statement.

Code:

```
if [ -f $1 ]
then
echo “It is a file”
```

```
else if [ -d $1 ]
then
echo “It is a directory”
else
echo “Invalid filename”
fi
fi
```

Save the file as file2.sh

Sample Input:
chmod 754 file2.sh
./file2.sh file1.sh
(Note: Here file1.sh is the command line argument.)

Sample Output:
It is a file

c. Write a shell script to input the marks of a student in 3 subjects and find his grade.

Procedure/Algorithm/Steps/Syntax:
Any expression can be calculated in shell programming using expr. The options for equal to is –eq, for greater than equal to is –ge, for less than equal to is –le, and so on.

Code:

```
echo “Enter marks in three subjects”
read m1
read m2
read m3
sum=`expr $m1+$m2+$m3`
per=`expr $sum/3`
if [ $per –ge 60 ]
then
echo “First division”
else if [ $per –ge 50 ]
then
echo “Second Division”
else if [ $per –ge 40 ]
then
echo “Third Division”
else
echo “Fail”
fi
fi
fi
```

Save the file as file1.sh

Sample Input:
chmod 754 file1.sh
./file1.sh

Sample Output:
Enter marks in three subjects
98
97
90
First division

Post Experiment Question:

1. Explain the if-else construct.
2. What is the difference between x and $x?
3. What is the difference between ‘’ and “”?
4. What is the need of expr?
5. Explain –ge, -eq, -le, etc?

EXPERIMENT NO: 7

Environment: Microsoft Windows/Linux

Objective: Shell scripts related to strings:

a. Write a shell script to input two strings from the user and determine whether they are same or not.

Procedure/Algorithm/Steps/Syntax:
Two strings can be compared using "=" sign

Code:
```
echo "Enter string1"
read s1
echo "Enter string2"
read s2
if [ s1 = s2 ]
then
echo "Strings are same"
else
echo "Strings do not match"
fi
```

Save the file as file1.sh

Sample Input:
```
chmod 754 file1.sh
./file1.sh
```

Sample Output:
```
Enter string1
University
Enter string2
University
Strings are same
```

b. Write a shell script to input a string from the user and determine its length.

Procedure/Algorithm/Steps/Syntax:
The length of a string can be determined using #.
When we write
len=$ {#s1}
the length of the string s1 gets stored in the len variable.

Code:
```
echo "Enter string"
read s1
len=${#s1}
echo "Length of the string is $len"
```

Save the file as file1.sh

Sample Input:
chmod 754 file1.sh
./file1.sh

Sample Output:
Enter string
GLA University
Length of the string is 14

c. Write a shell script to input two strings from the user and find the occurrences of string2 in string 1.

Procedure/Algorithm/Steps/Syntax:
The patterns can be matched using grep command. Similarly, we can match the strings within each other using grep command.
The –o option is for matching and the –w option is for a complete word only.

Code:
```
echo “Enter string1”
read s1
echo “Enter string2”
read s2
grep –o –w “$s2” <<<  “$s1” | wc -l
```

Save the file as file1.sh

Sample Input:
chmod 754 file1.sh
./file1.sh

Sample Output:
Enter string1
Jitendra is ten years old
Enter string2
ten
1

Post Experiment Questions:
1. How can you calculate the length of the string?
2. Is there any other method to calculate length of the string? Explain.
3. What are strings? How are strings managed in UNIX?
4. How can you compare 2 strings?
5. How can a grep command used to search in a file?
6. What are the various uses of grep?

EXPERIMENT NO: 8

Environment: Microsoft Windows/Linux

Objective: Shell scripts using pipes:

a. Write a shell script to input the name of a file as command line argument and display the number of characters, words and lines in the file.

Procedure/Algorithm/Steps/Syntax:

The entered name is a file or not can be checked by the option –f. The grep command helps us to find the number of lines, characters and words in the file.

Code:

```
if [ -f $1 ]
then
w=`cat $1 | wc –w`
c=`cat $1 | wc –c`
l=`cat $1 | wc –l`
echo “Number of words=$w”
echo “Number of lines=$l”
echo “Number of characters=$c”
else
echo “Invalid filename”
fi
```

Save the file as file2.sh

Sample Input:

chmod 754 file2.sh
./file2.sh file1.sh
(Note: Here file1.sh is the command line argument.)

Sample Output:
Number of words=25
Number of lines=2
Number of characters=243

b. Write a shell script to display a list of directories within the current directory and how much space they consume, sorted from the largest to the smallest.

Procedure/Algorithm/Steps/Syntax:

The du command is used to find the disk usage. The sort command is used for sorting. The –n option with sort command is for numeric sort. The –r option with sort command is used to sort in reverse order.

Code:

```
du -sh */ | sort -nr
```

Save the file as file2.sh

Sample Input:

chmod 754 file2.sh
./file2.sh

Sample Output:
List of directories sorted from largest to smallest.

Post Experiment Question:

1. What are the options for checking a directory and a file?
2. How many types of files are there in UNIX? Explain.
3. What is the use of cat in this program?
4. Explain the line w=`cat $1 | wc –w`
5. Explain the du command.
6. How does sort command sorts in reverse order?

EXPERIMENT NO: 9

Environment: Microsoft Windows/Linux

Objective: Shell scripts with loop statements:

a. **Write a shell script that inputs a number from the user and prints its table on the screen.**

Procedure/Algorithm/Steps/Syntax:

The frequently used two loops in unix shell scripts are while loop and for loop.

Code:

```
echo "Enter the value of  n:"
read n
for((i=1;i<=10;i++))
do
echo " $n * $i = `expr $n \* $i`"
done
```

Save the file as file2.sh

Sample Input:

```
chmod 754 file2.sh
./file2.sh
```

Sample Output:

```
Enter the value of n:
5
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

b. **Write a shell script to implement a timer.**

Code:

```
ch=1
while [ $ch -eq 1 ]
do
echo "Enter a Timer Limit"
read n
```

```
for (( i=n; i>0; i--)); do
  sleep 1 &
  printf "Timer  $i \r"
  wait
done
echo "Time Up!"
echo "Want a new Timer? Press 1 to continue 0 to exit!"
read ch
done
```

Save the file as file2.sh

Sample Input:

```
chmod 754 file2.sh
./file2.sh
```

Sample Output:

```
Enter the timer limit
5
(Timer runs for 5 seconds)
Time Up
Want a new Timer? Press 1 to continue 0 to exit!
0
```

Post Experiment Question:

1. What are the syntax of for and while loop?
2. Explain the syntax of expr command for multiplication of two numbers.

EXPERIMENT NO: 10

Environment: Microsoft Windows/Linux

Tools/ Language: Telnet

Objective: Implement the basic IPCS commands used in UNIX.

Procedure/Algorithm/Steps/Syntax:
IPC stands for Inter-process Communication. This technique allows the processes to communicate with each another.
We can request the kernel to allocate the space which can be used to communicate between processes. The process can also communicate by having a file accessible to both the processes. Processes can open, and read/write the file, which requires lot of I/O operation that consumes time.

Different Types of IPCS

There are various IPC's which allows a process to communicate with another process, either in the same computer or different computer in the same network.

1. **Pipes** – Provides a way for processes to communicate with each another by exchanging messages. Named pipes provide a way for processes running on different computer systems to communicate over the network.
2. **Shared Memory** – Processes can exchange values in the shared memory. One process will create a portion of memory which other process can access.
3. **Message Queue** – It is a structured and ordered list of memory segments where processes store or retrieve data.
4. **Semaphores** – Provides a synchronizing mechanism for processes that are accessing the same resource. No data is passed with a semaphore; it simply coordinates access to shared resources.

Code:
```
ipcs –a
ipcs –q
ipcs –s
ipcs –m
ipcs -q -i 32768
```

Sample Input:

1. ipcs –a
2. ipcs –q
3. ipcs –s
4. ipcs –m
5. ipcs -q -i 32768

Sample Output:

1. ipcs command with -a option lists all the IPC facilities which has read access for the current process. It provides details about message queue, semaphore and shared memory.

2. ipcs with option -q, lists only message queues for which the current process has read access.
3. ipcs -s option is used to list the accessible semaphores.
4. ipcs -m option with ipcs command lists the shared memories.
5. ipcs -i option provides detailed information about an ipc facility. Option -i with -q provides information about a particular message queue. Option -i with -s provides semaphore details. Option -i with -m provides details about a shared memory.

Post Experiment Question:

1. What do you mean by IPC? What are the various categories that lie within it?
2. Explain the various options used with IPCS.