LABORATORY MANUAL

FOR

IV -SEMESTER COMPUTER SCIENCE& ENGINEERING

Linux Shell Programming Lab- 4CS4-24

INTERNAL MARKS: 30

EXTERNAL MARKS: 20

Department of Computer Science and Engineering

GLOBAL INSTITUTE OF TECHNOLOGY, JAIPUR

Scheme as per RTU

Subject Code	Name of Subject	Exam Hrs.	L	T	P	M.M Sessional/ Mid Term	M.M. End Term	Total M.M.
4CS4-24	Linux Shell Prog. Lab	2	-	-	2	30	20	50

Assessment criteria

A. Internal Assessment: 30

In continuous evaluation system of the university, a student is evaluated throughout semester. His/her performance in the lab, attendance, practical knowledge, problem solving skill, written work in practical file and behavior are main criteria to evaluate student performance. Apart from that a lab quiz will be organize to see program programming skill and knowledge about the proposed subject.

B. External Assessment: 20

At the end of the semester a lab examination will be scheduled to check overall programming skill, in which student will need to solve 2 programming problems in time span of 3 hours.

C. Total Marks: 30+20=50

SYLLABUS AS PER RTU

1. Use of basic Unix Shell Commands: ls, mkdir, rmdir, cd, cat, banner, touch, file, wc, sort, cut, grep, dd, dfspace, du, ulimit.
2. Commands related to inode, I/O redirection, piping, process control commands, mails.
3. Shell Programming: shell script exercise based on following:
   - **a.** Interactive shell script
   - **b.** Positional parameters
   - **c.** Arithmetic
   - **d.** If-then-fi, if-then-else-fi, nested if-else
   - **e.** Logical operators
   - **f.** Else + if equals elif, case structure
   - **g.** While ,for loop
   - **h.** Meta characters
4. Write a shell script to create a file in $USER /class/batch directory. Follow the Instructions
   - Input a page profile to yourself, copy it into other existing file
   - Start printing file at certain line
   - Print all the difference between two file, copy the two files at $USER/CSC/2007 directory.
   - Print lines matching certain word pattern.
5. Write shell script for-
   - Showing the count of users logged in
   - Printing Column list of files in your home directory.
   - Listing your job with below normal priority
   - Continue running your job after logging out.
6. Write a shell script to change date format. Show the time taken in execution of this script.
7. Write a shell script to print file names in directory showing date of creation & serial no. of file.
8. Write a shell script to count lines, words & characters in its input. (do not use wc)
9. Write a shell script to print end of a Glossary file in reverse order using array.
10. Write a shell script to check whether Ram logged in, continue checking further after every 30 seconds till success.
11. Write a shell script to compute GCD & LCM of two numbers.

12. Write a shell script to find whether a given number is prime.

LIST OF SHELL PROGRAMMING LAB EXPERIMENTS:

1. Use of basic Unix Shell Commands: ls, mkdir, rmdir, cd, cat, banner, touch, file, wc, sort, cut, grep, dd, dfspace, du, ulimit.
2. Commands related to inode, I/O redirection, piping, process control commands, mails.
3. Shell Programming: shell script exercise based on following:
    - **Interactive shell script**
    - **Positional parameters**
    - **Arithmetic**
    - **If-then-fi, if-then-else-fi, nested if-else**
    - **Logical operators**
    - **Else + if equals elif, case structure**
    - **While ,for loop**
    - **Meta characters**
4. Write a shell script to create a file in $USER /class/batch directory. Follow the Instructions
    - Input a page profile to yourself, copy it into other existing file
    - Start printing file at certain line
    - Print all the difference between two file, copy the two files at $USER/CSC/2007 directory.
    - Print lines matching certain word pattern.
5. Write shell script for-
    - Showing the count of users logged in
    - Printing Column list of files in your home directory.
    - Listing your job with below normal priority
    - Continue running your job after logging out.
6. Write a shell script to change date format. Show the time taken in execution of this script
7. Write a shell script to print file names in directory showing date of creation & serial no. of file.
8. Write a shell script to count lines, words & characters in its input. (do not use wc)
9. Write a shell script to print end of a Glossary file in reverse order using array.
10. Write a shell script to check whether Ram logged in, continue checking further after every 30 seconds till success.
11. Write a shell script to compute GCD & LCM of two numbers.
12. Write a shell script to find whether a given number is prime.

BEYOND CURRICULUM:

- **a.** Introduction of Linux
- **b.** Shell script to perform database operations for student data like view, add and delete records in Unix / Linux.

Subject: Linux Shell Programming Lab **Subject Code: 4CS4-24**
Branch: Computer Science &Engineering **Year:** 2 **Semester:** 4th
Mid-Term Marks:30 **End Term Marks:**20

Lab Plan

S. No.	Experiments & Necessary background	No. of Labs
1	Use of basic Unix Shell Commands: ls, mkdir, rmdir, cd, cat, banner, touch, file, wc, sort, cut, grep, dd, dfspace, du, ulimit	
2	Commands related to inode, I/O redirection, piping, process control commands, mails.	
3	Shell Programming: shell script exercise based on following: • **Interactive shell script** • **Positional parameters** • **Arithmetic** • **If-then-fi, if-then-else-fi, nested if-else** • **Logical operators** • **Else + if equals elif, case structure** • **While ,for loop** • **Meta characters**	
4	Write a shell script to create a file in $USER /class/batch directory. Follow the Instructions • Input a page profile to yourself, copy it into other existing file • Start printing file at certain line • Print all the difference between two file, copy the two files at $USER/CSC/2007 directory.	
5	Write shell script for- • Showing the count of users logged in • Printing Column list of files in your home directory. • Listing your job with below normal priority • Continue running your job after logging out	
6	Write a shell script to change date format. Show the time taken in execution of this script	
7	Write a shell script to print file names in directory showing date of creation & serial no. of file.	
8	Write a shell script to count lines, words & characters in its input. (do not use wc)	
9	Write a shell script to print end of a Glossary file in reverse order using array	
10	Write a shell script to check whether Ram logged in, continue checking further after every 30 seconds till success.	
11	Write a shell script to compute GCD & LCM of two numbers.	
12	Write a shell script to find whether a given number is prime	

BEYOND CURRICULUM		
1	Introduction of Linux	
2	Shell script to perform database operations for student data like view, add and delete records in Unix / Linux	

Objectives of the laboratory:

- Mention the features of UNIX;
- Recognize, understand and make use of various UNIX commands;
- Gain hands on experience of UNIX commands and shell programs;
- Feel more confident about writing the shell scripts and shell programs;
- Apply the concepts that have been covered in this manual, and
- Know the alternate ways of providing the solutions to the given practical exercises and problems.

Duties before the lab starts:

Students are required to submit the lab file before the lab starts. Students missing the lab file submission will not be accepted to the lab.

Tools used in lab:

Operating System- Linux

Rules of behavior in the laboratory:

a). Every laboratory sessions begins SHARP at the specified time in the schedule.
b). Each lab session is two hours long. Students are advised to bring their practical file
c). Please bring your identity cards with you.
d). Students must have their own copies of the laboratory manual.
e). Food, drinks and cell phone are not allowed inside the laboratory.

How to write Source program in practical file:

a). Every program should be beginning from new fresh page.
b). Students are supposed to follow the prescribed format to write program in practical file.
c). Index page should be the first page in the practical file and write the complete objective the index, what you have written as the aim in the program.

Experiment No. 1

1 AIM: (A) Explain the following commands:

1. clear
2. cal
3. who
4. date
5. mkdir
6. rm
7. cat
8. cd
9. cp
10. grep
11. ls
12. mv
13. rm
14. rmdir

2 TOOLS/APPARATUS: Linux operating system.

3 STANDARD PROCEDURES:

3.1 Analyzing the Problem:

- Start the Linux and enter the user name and password.
- Now write startx and after that open the terminal.
- At the terminal try the different commands and see the output.

3.2 Designing the Solution:

- At the terminal first perform the command CAL without and with the different options available for it.
- Like $ cal and then enter. The calendar will be displayed at the terminal.
- $ cal –m and then enter. In the calendar Monday will be displayed as the first day of the week.
- Same way perform the other commands like CLEAR, WHO, DATE, MKDIR, RM etc.

3.3 Implementing the Solution:

3.3.1 Writing Source Code:

1) CAL:

At the terminal write the following:

 [user1@com]$ cal
 [user1@com]$ cal -m
 [user1@com]$ cal -j
 [user1@com]$ cal –y

2) CLEAR:

At the terminal write the following:

 [user1@com]$ clear

3) WHO:

At the terminal write the following:

- [user1@com]$ who
- [user1@com]$ who -q
- [user1@com]$ who -H
- [user1@com]$ who –m

4) DATE:

At the terminal write the following:

- [user1@com]$ date
- [user1@com]$ date –d "2 days ago"
- [user1@com]$ date +%D
- [user1@com]$ date +%d
- [user1@com]$ date +%d%m%h

5) MKDIR and RM:

At the terminal write the following:

- [user1@com]$ cd Desktop/
- [user1@com]$ ls
- [user1@com]$ cd newfiles/
- [user1@com]$ ls
- [user1@com]$ mkdir newfile1
- [user1@com]$ ls
- [user1@com]$ rm Sum_Of_Digits.txt
- [user1@com]$ ls

6)cat

catallows you to read multiple files and then print them out. You can combine files by using the > operator and append files by using >>.

Syntax: ***cat [argument] [specific file]***

Example:

cat abc.txt

If you want to append three files (abc.txt, def.txt, xyz.txt), give the command as,
cat abc.txt def.txt xyz.txt > all

***7)*cd, chdir**

cd (or chdir) stands for "change directory". This command is the key command to move around your file structure.

Syntax: ***cd [name of directory you want to move to]***

When changing directories, start with / and then type the complete file path, like
cd /vvs/abc/xyz

8)cp

The ***cp*** command copies files or directories from one place to another. You can copy a set of files to another file, or copy one or more files under the same name in a directory. If the destination of the file you want to copy is an existing file, then the

existing file is overwritten. If the destination is an existing directory, then the file is copied into that directory.

Syntax: ***cp [options] file1 file2***

If you want to copy the file ***favourites.html*** into the directory called ***laksh***, you give the command as:

cp favourites.html /vvs/laksh/

A handy option to use with ***cp*** is ***-r***. This recursively copies a particular directory and all of its contents to the specified directory, so you won't have to copy one file at a time.

9)grep

The ***grep*** command searches a file or files for lines that match a provided regular expression ("grep" comes from a command meaning to **g**lobally search for a **r**egular **e**xpression and then **p**rint the found matches).

Syntax: ***grep [options] regular expression [files]***

To exit this command, type 0 if lines have matched, 1 if no lines match, and 2 for errors. This is very useful if you need to match things in several files. If you wanted to find out which files in our ***vvs***directory contained the word "***mca***" you could use ***grep*** to search the directory and match those files with that word. All that you have to do is give the command as shown:

grep 'mca' /vvs/*

The * used in this example is called a meta-character, and it represents matching zero or more of the preceding characters. In this example, it is used to mean "all files and directories in this directory". So, ***grep*** will search all the files and directories in ***vvs***and tell you which files contain "***mca"***.

10)ls

lswill list all the files in the current directory. If one or more files are given, ***ls*** will display the files contained within "name" or list all the files with the same name as "name". The files can be displayed in a variety of formats using various options.

Syntax: ***ls [options] [names]***

lsis a command you'll end up using all the time. It simply stands for list. If you are in a directory and you want to know what files and directories are inside that directory, type ***ls***. Sometimes the list of files is very long and it flies past your screen so quickly you miss the file you want. To overcome this problem give the command as shown below:

ls | more

The character | (called pipe) is typed by using shift and the \ key. | ***more*** will show as many files as will fit on your screen, and then display a highlighted ***"more"*** at the bottom. If you want to see the next screen, hit enter (for moving one line at a time) or the spacebar (to move a screen at a time). **|** ***more***can be used anytime you wish to view the output of a command in this way.

A useful option to use with ***ls*** command is ***-l***. This will list the files and directories in a long format. This means it will display the permissions (see chmod), owners, group, size, date and time the file was last modified, and the filename.

drwxrwxr-xvvs staff 512 Apr 5 09:34 sridhar.txt
-rwx-rw-r-- vvs staff 4233 Apr 1 10:20 resume.txt
-rwx-r--r-- vvs staff 4122 Apr 1 12:01 favourites.html
There are several other options that can be used to modify the ls command, and many of these options can be combined. ***-a*** will list all files in a directory, including those files normally hidden. ***-F*** will flag filenames by putting / on directories, @ on symbolic links, and * on executable files.

11) mv

mv moves files and directories. It can also be used to rename files or directories.
Syntax: mv [options] source target
If you wanted to rename vvs.txt to vsv.txt, you should give the command as:
mv vvs.txt vsv.txt
After executing this command, vvs.txt would no longer exist, but a file with name vsv.txt would now exist with the same contents.

12)rm

rmremoves or deletes files from a directory.

Syntax: ***rm [options] files***

In order to remove a file, you must have write permission to the directory where the file is located. While removing a which does't have write permission on, a prompt will come up asking you whether or not you wish to override the write protection.

The ***-r*** option is very handy and very dangerous. ***-r*** can be used to remove a directory and all its contents. If you use the ***-i*** option, you can possibly catch some disastrous mistakes because it'll ask you to confirm whether you really want to remove a file before going ahead and doing it.

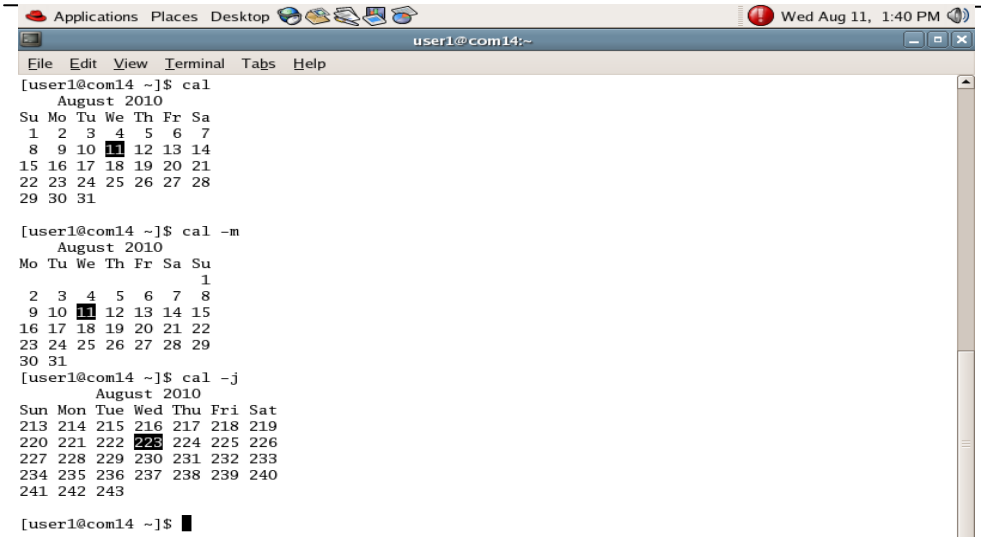
13)rmdir

rmdirallows you to remove or delete directories but not their contents. A directory must be empty in order to remove it using this command.

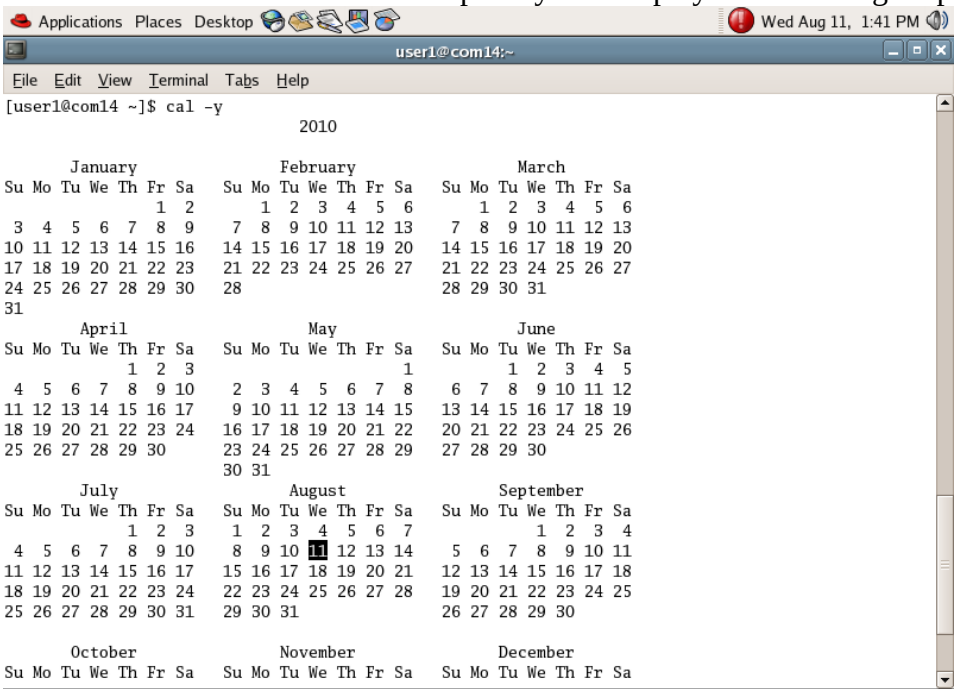
Syntax: ***rmdir [options] directories***
If you wish to remove a directory and all its contents, you should use rm -r.

Compilation /Running and Debugging the Solution:

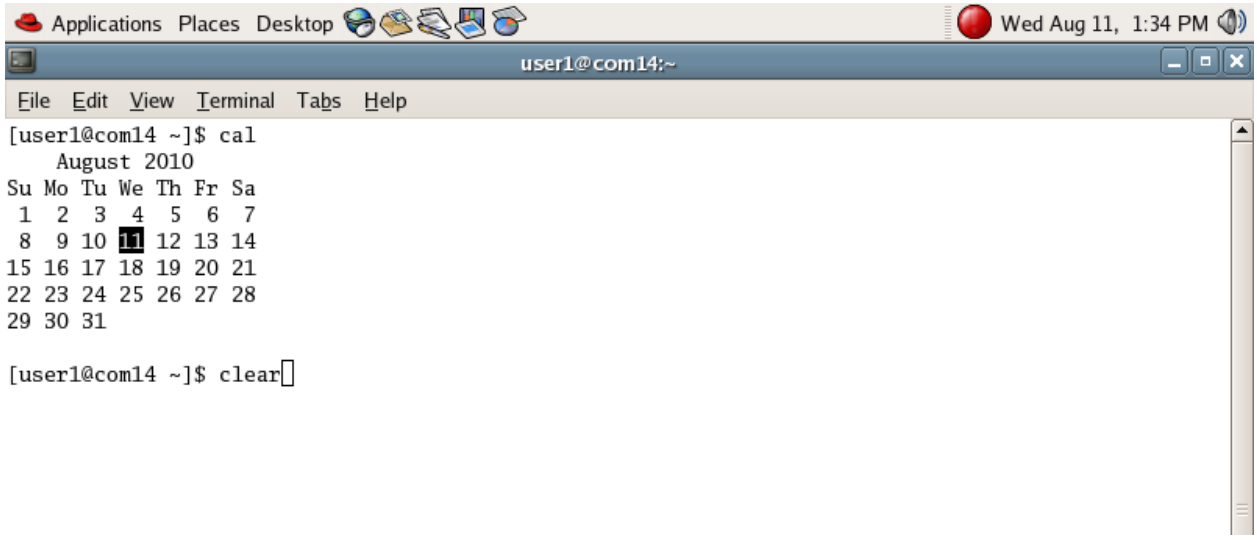
- The code written above will display the following output.
  For the first command CAL the output is like this:

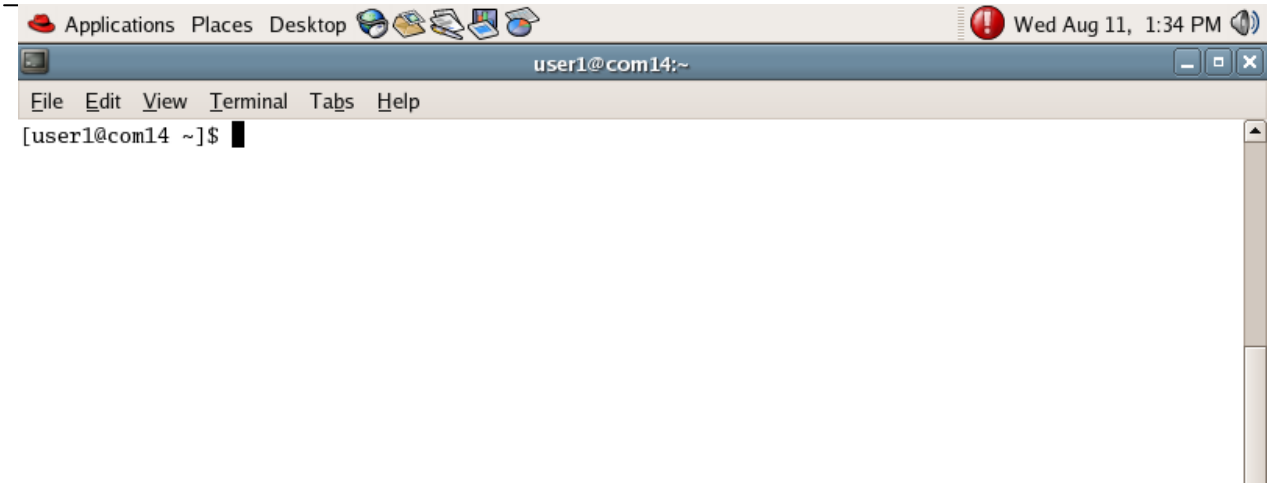


The cal command with the option y will display the following output.

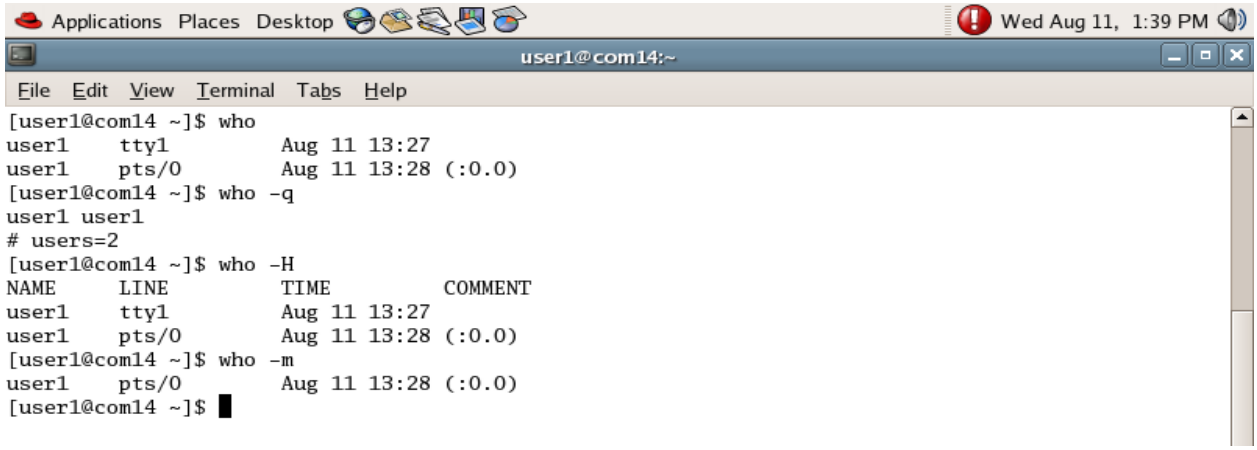


For the second command CLEAR :

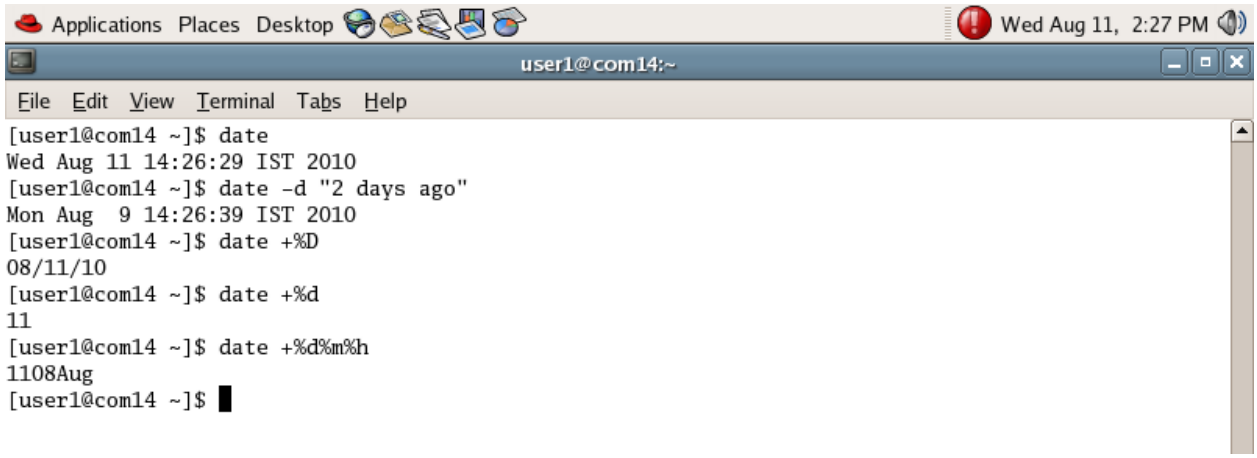




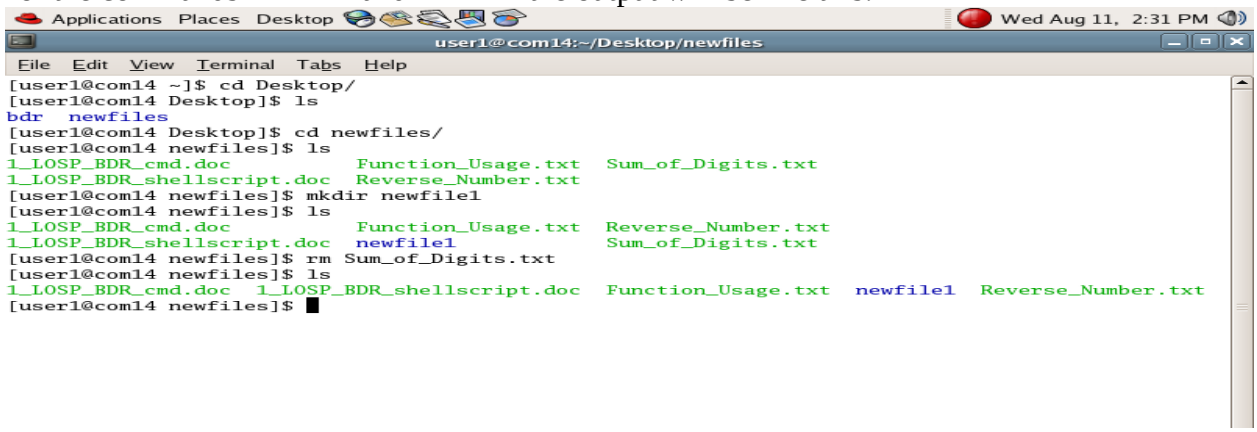
For the third command WHO :



For the command DATE:



For the commands MKDIR and RMDIR the output will be like this:



Testing the Solution:

- All the commands will display the output based on it and the options given to that
  command.
- If we are giving a command and the option to that command then that option must be
  of that command only otherwise will display the error.

Conclusions :

Using this we can run different command and see the output.

Lab 2

1. Aim: Commands related to inode, I/O redirection and piping, mail, xargs, export, set-unset, source, ps, kill, jobs.
2. Software Used: Operating System: Linux
3. Source Code/experiment description:

mail:
Send or read e-mail messages.
This stripped-down command-line mail client works fine as a command embedded in a script.

Example: A script that mails itself

```
#!/bin/sh
# self-mailer.sh: Self-mailing script
adr=${1:-`whoami`} # Default to current user, if not specified.
# Typing 'self-mailer.sh wiseguy@superdupergenius.com'
#+ sends this script to that addressee.
# Just 'self-mailer.sh' (no argument) sends the script
#+ to the person invoking it, for example, bozo@localhost.localdomain.
#
# For more on the ${parameter:-default} construct,
#+ see the "Parameter Substitution" section
#+ of the "Variables Revisited" chapter.
#
============================================================================
=======
cat $0 | mail -s "Script \"`basename $0`\" has mailed itself to you." "$adr"
#
============================================================================
=======
# ------------------------------------------
# Greetings from the self-mailing script.
# A mischievous person has run this script,
#+ which has caused it to mail itself to you.
# Apparently, some people have nothing better
#+ to do with their time.
# ------------------------------------------
echo "At `date`, script \"`basename $0`\" mailed to "$adr"."
exit 0
```

inode:

Example: Deleting a file by its inode number

```
#!/bin/bash
# idelete.sh: Deleting a file by its inode number.
# This is useful when a filename starts with an illegal character,
```

```
#+ such as ?or -.
ARGCOUNT=1 # Filename arg must be passed to script.
E_WRONGARGS=70
E_FILE_NOT_EXIST=71
E_CHANGED_MIND=72
if [ $# -ne "$ARGCOUNT" ]
then
echo "Usage: `basename $0` filename"
exit $E_WRONGARGS
fi
if [ ! -e "$1" ]
then
echo "File \""$1"\" does not exist."
exit $E_FILE_NOT_EXIST
fi
inum=`ls -i | grep "$1" | awk '{print $1}'`
# inum = inode (index node) number of file
# ---------------------------------------------------------------------
# Every file has an inode, a record that holds its physical address info.
# ---------------------------------------------------------------------
echo; echo -n "Are you absolutely sure you want to delete \"$1\" (y/n)? "
# The '-v' option to 'rm' also asks this.
read answer
case "$answer" in
[nN]) echo "Changed your mind, huh?"
exit $E_CHANGED_MIND
;;
*) echo "Deleting file \"$1\".";;
esac
find . -inum $inum -exec rm {} \;
# ^^
# Curly brackets are placeholder
#+ for text output by "find."
echo "File "\"$1"\" deleted!"
exit 0
```

Piping:

Example: Piping the output of echo to a read

```
#!/bin/bash
# badread.sh:
# Attempting to use 'echo and 'read'
#+ to assign variables non-interactively.
a=aaa
b=bbb
c=ccc
```

```
echo "one two three" | read a b c
# Try to reassign a, b, and c.
echo
echo "a = $a" # a = aaa
echo "b = $b" # b = bbb
echo "c = $c" # c = ccc
# Reassignment failed.
# ------------------------------
# Try the following alternative.
var=`echo "one two three"`
set -- $var
a=$1; b=$2; c=$3
echo "-------"
echo "a = $a" # a = one
echo "b = $b" # b = two
echo "c = $c" # c = three

a=aaa # Starting all over again.
b=bbb
c=ccc
echo; echo
echo "one two three" | ( read a b c;
echo "Inside subshell: "; echo "a = $a"; echo "b = $b"; echo "c = $c" )
# a = one
# b = two
# c = three
echo "-----------------"
echo "Outside subshell: "
echo "a = $a" # a = aaa
echo "b = $b" # b = bbb
echo "c = $c" # c = ccc
echo
exit 0
```

I/O Redirection:

Example1: Redirecting stdin using exec

```
#!/bin/bash
# Redirectingstdin using 'exec'.
exec 6<&0 # Link file descriptor #6 with stdin.
# Saves stdin.
exec< data-file # stdin replaced by file "data-file"
read a1 # Reads first line of file "data-file".
read a2 # Reads second line of file "data-file."
echo
echo "Following lines read from file."
```

```
echo "-------------------------------"
echo $a1
echo $a2
echo; echo; echo
exec 0<&6 6<&-
# Now restore stdin from fd #6, where it had been saved,
#+ and close fd #6 ( 6<&- ) to free it for other processes to use.
#
# <&6 6<&- also works.
echo -n "Enter data "
read b1 # Now "read" functions as expected, reading from normal stdin.
echo "Input read from stdin."
echo "----------------------"
echo "b1 = $b1"
echo
exit 0
```

Example2: Redirecting stdout using exec

```
#!/bin/bash
# reassign-stdout.sh
LOGFILE=logfile.txt
exec 6>&1 # Link file descriptor #6 with stdout.
# Saves stdout.
exec> $LOGFILE # stdout replaced with file "logfile.txt".
# ----------------------------------------------------------- #
# All output from commands in this block sent to file $LOGFILE.
echo -n "Logfile: "
date
echo "-------------------------------------"
echo
echo "Output of \"ls -al\" command"
echo
ls -al
echo; echo
echo "Output of \"df\" command"
echo
df
# ----------------------------------------------------------- #
exec 1>&6 6>&- # Restore stdout and close file descriptor #6.
echo
echo "== stdout now restored to default == "
echo
ls -al
echo
exit 0
```

xargs:

A filter for feeding arguments to a command, and also a tool for assembling the commands themselves. It breaks a data stream into small enough chunks for filters and commands to process. Consider it as a powerful replacement for backquotes. In situations where command substitution fails with a too many arguments error, substituting xargs often works. [65] Normally, xargs reads from stdin or from a pipe, but it can also be given the output of a file.

The default command for xargs is echo. This means that input piped to xargs may have linefeeds and other whitespace characters stripped out.

bash$ ls -l
total 0
-rw-rw-r-- 1 bozo bozo 0 Jan 29 23:58 file1
-rw-rw-r-- 1 bozo bozo 0 Jan 29 23:58 file2
bash$ ls -l | xargs
total 0 -rw-rw-r-- 1 bozo bozo 0 Jan 29 23:58 file1 -rw-rw-r-- 1 bozo bozo 0 Jan...
bash$ find ~/mail -type f | xargs grep "Linux"
./misc:User-Agent: slrn/0.9.8.1 (Linux)
./sent-mail-jul-2005: hosted by the Linux Documentation Project.
./sent-mail-jul-2005: (Linux Documentation Project Site, rtf version)
./sent-mail-jul-2005: Subject: Criticism of Bozo's Windows/Linux article
./sent-mail-jul-2005: while mentioning that the Linux ext2/ext3 filesystem
. . .

ls | xargs -p -l gzipgzips every file in current directory, one at a time, prompting before each operation.

Note that xargs processes the arguments passed to it sequentially, one at a time.

bash$ find /usr/bin | xargs file
/usr/bin: directory
/usr/bin/foomatic-ppd-options: perl script text executable
. . .

An interesting xargs option is -n NN, which limits to NN the number of arguments passed.

ls | xargs -n 8 echo lists the files in the current directory in 8 columns.

Another useful option is -0, in combination with find -print0 or grep -lZ. This allows handling arguments containing
whitespace or quotes.

find / -type f -print0 | xargs -0 grep -liwZ GUI | xargs
-0 rm -f
grep-rliwZ GUI / | xargs -0 rm -f

Set-Unset:

set

The set command changes the value of internal script variables/options. One use for this is to toggle option flags
which help determine the behavior of the script. Another application for it is to reset the positional parameters that a script sees as the result of a command (set `command`). The script can then parse the fields of the command output.

Example: Using set with positional parameters

```
#!/bin/bash
# ex34.sh
# Script "set-test"
# Invoke this script with three command-line parameters,
# for example, "sh ex34.sh one two three".
echo
echo "Positional parameters before set \`uname -a\` :"
echo "Command-line argument #1 = $1"
echo "Command-line argument #2 = $2"
echo "Command-line argument #3 = $3"
set `uname -a` # Sets the positional parameters to the output
# of the command `uname -a`
echo
echo +++++
echo $_ # +++++
# Flags set in script.
echo $- # hB
# Anomalous behavior?
echo
echo "Positional parameters after set \`uname -a\` :"
# $1, $2, $3, etc. reinitialized to result of `uname -a`
echo "Field #1 of 'uname -a' = $1"
echo "Field #2 of 'uname -a' = $2"
echo "Field #3 of 'uname -a' = $3"
echo \#\#\#
echo $_ # ###
echo
exit 0
```

unset

The unset command deletes a shell variable, effectively setting it to null. Note that this command does not affect positional parameters.

```
bash$ unset PATH
bash$ echo $PATH
bash$
```

Example: "Unsetting" a variable

```
#!/bin/bash
# unset.sh: Unsetting a variable.
variable=hello # Initialized.
echo "variable = $variable"
unset variable # Unset.
# In this particular context,
#+ same effect as: variable=
echo "(unset) variable = $variable" # $variable is null.
if [ -z "$variable" ] # Try a string-length test.
then
```

```
echo "\$variable has zero length."
fi
exit 0
```

export:

The export command makes available variables to all child processes of the running script or shell. One important use of the export command is in startup files, to initialize and make accessible environmental variables to subsequent user processes.

Note: Unfortunately, there is no way to export variables back to the parent process, to the process that called or invoked the script or shell.

Example: Using export to pass a variable to an embedded awk script

```
#!/bin/bash
# Yet another version of the "column totaler" script (col-totaler.sh)
#+ that adds up a specified column (of numbers) in the target file.
# This uses the environment to pass a script variable to 'awk' . . .
#+ and places the awk script in a variable.
ARGS=2
E_WRONGARGS=85
if [ $# -ne "$ARGS" ] # Check for proper number of command-line args.
then
echo "Usage: `basename $0` filename column-number"
exit $E_WRONGARGS
fi
filename=$1
column_number=$2
#===== Same as original script, up to this point =====#
exportcolumn_number
# Export column number to environment, so it's available for retrieval.
# ----------------------------------------------
awkscript='{ total += $ENVIRON["column_number"] }
END { print total }'
# Yes, a variable can hold an awk script.
# ----------------------------------------------
# Now, run the awk script.
awk "$awkscript" "$filename"
# Thanks, Stephane Chazelas.
exit 0
```

It is possible to initialize and export variables in the same operation, as in export var1=xxx.

source, . (dot command):

This command, when invoked from the command-line, executes a script. Within a script, a source file-name loads the file file-name. Sourcing a file (dot-command) imports code into the script, appending to the script (same effect as the #include directive in a C program). The net result is the same as if the "sourced" lines of code were physically present in the body of the script. This is useful in situations when multiple scripts use a common data file or function library.

Example: "Including" a data file
#!/bin/bash
. data-file # Load a data file.
Same effect as "source data-file", but more portable.
The file "data-file" must be present in current working directory,
#+ since it is referred to by its 'basename'.
Now, reference some data from that file.
echo "variable1 (from data-file) = $variable1"
Advanced Bash-Scripting Guide
Chapter 15. Internal Commands and Builtins 198
echo "variable3 (from data-file) = $variable3"
let "sum = $variable2 + $variable4"
echo "Sum of variable2 + variable4 (from data-file) = $sum"
echo "message1 (from data-file) is \"$message1\""
Note: escaped quotes
Print_messageThis is the message-print function in the data-file.
exit 0
File data-file for above Example Must be present in same directory.

ps:
Process Statistics: lists currently executing processes by owner and PID (process ID). This is usually invoked with ax or aux options, and may be piped to grep or sed to search for a specific process.

Example:
bash$ ps ax | grep sendmail
295 ? S 0:00 sendmail: accepting connections on port 25
To display system processes in graphical "tree" format: psafjx or ps ax --forest.

kill:
Forcibly terminate a process by sending it an appropriate terminate signal.

Example: A script that kills itself
#!/bin/bash
self-destruct.sh
kill $$ # Script kills its own process here.
Recall that "$$" is the script's PID.
echo "This line will not echo."
Instead, the shell sends a "Terminated" message to stdout.
exit 0 # Normal exit? No!
After this script terminates prematurely,
#+ what exit status does it return?

```
#
# sh self-destruct.sh
# echo $?
# 143
#
# 143 = 128 + 15
# TERM signal
```

kill -l lists all the signals (as does the file /usr/include/asm/signal.h). A kill -9 is a sure kill, which will usually terminate a process that stubbornly refuses to die with a plain kill. Sometimes, a kill -15 works. A zombie process, that is, a child process that has terminated, but that the parent process has not (yet) killed, cannot be killed by a logged-on user -- you can't kill something that is already dead -- but init will generally clean it up sooner or later.

Jobs:

Lists the jobs running in the background, giving the job number. Not as useful as ps.

It is all too easy to confuse jobs and processes. Certain builtins, such as kill, disown, and wait accept either a job number or a process number as an argument. The fg, bg and jobs commands accept only a job number.

Example:

```
bash$ sleep 100 &
[1] 1384
bash $ jobs
[1]+ Running sleep 100 &
```

"1" is the job number (jobs are maintained by the current shell). "1384" is the PID or process ID number (processes are maintained by the system). To kill this job/process, either a kill %1 or a kill 1384 works.

4. Conclusion: In this experiment student learn various commands for shell scripting.

Lab 3

1. **AIM:** Shell Programming: shell script exercise based on following:
   - **Interactive shell script**
   - **Positional parameters**
   - **Arithmetic**
   - **If-then-fi, if-then-else-fi, nested if-else**
   - **Logical operators**
   - **Else + if equals elif, case structure**
   - **While ,for loop**
   - **Meta characters**

2. SOFTWARE USED: Operating System: Linux

3. SOURCE CODE:

If-then-fi, if-then-else-fi, nested if-else-

```
if [ $# -ne 1 ]
then
echo "Usage - $0 file-name" exit 1 fi
if [ -f $1
then
echo "$1 file exist"
else
echo "Sorry, $1 file does not exist"
fi
```

Logical operators-
integer comparison
-eq
is equal to
if ["$a" -eq "$b"]
-ne
is not equal to
if ["$a" -ne "$b"]
-gt
is greater than
if ["$a" -gt "$b"]
-ge
is greater than or equal to
if ["$a" -ge "$b"]
-lt
is less than
if ["$a" -lt "$b"]

-le
is less than or equal to
if ["$a" -le "$b"]
<
is less than (within double parentheses)
(("$a" < "$b"))
<=
is less than or equal to (within double parentheses)
(("$a" <= "$b"))
>
is greater than (within double parentheses)
(("$a" > "$b"))
>=
is greater than or equal to (within double parentheses)
(("$a" >= "$b"))
string comparison
=
is equal to
if ["$a" = "$b"]
==
is equal to
if ["$a" == "$b"]
This is a synonym for =.

```
1 [[ $a == z* ]]    # true if $a starts with an "z" (pattern matching)
2 [[ $a == "z*" ]]  # true if $a is equal to z*
3
4 [ $a == z* ]      # file globing and word splitting take place
5 [ "$a" == "z*" ]  # true if $a is equal to z*
6
7 # Thanks, S.C.
```

!=
is not equal to
if ["$a" != "$b"]
This operator uses pattern matching within a construct.
<
is less than, in ASCII alphabetical order
if [["$a" < "$b"]]
if ["$a" \< "$b"]
Note that the "<" needs to be escaped within a [ ] construct.
>
is greater than, in ASCII alphabetical order
if [["$a" > "$b"]]
if ["$a" \> "$b"]
Note that the ">" needs to be escaped within a [ ] construct.
See Example 26-6 for an application of this comparison operator.

-z
string is "null", that is, has zero length
-n
string is not "null".

Else + if equals elif, case structure

Shell Program to find Largest of Three Numbers

elif

```
clear
echo "Enter first number: "
read a
echo "Enter second number: "
read b
echo "Enter third number: "
read c
if [ $a -gt $b ] && [ $a -gt $c ]
then
echo "$a is greater"
elif [ $b -gt $a ] && [ $b -gt $c ]
then
echo "$b is greater"
elif [ $c -gt $a ] && [ $c -gt $b ]
then
echo "$c is greater"
fi
```

case structure

```
if test $# = 3
then
case $2 in
+) let z=$1+$3;;
-) let z=$1-$3;;
/) let z=$1/$3;; x|
X) let z=$1*$3;; *) echo Warning - $2 invalid operator, only +,-,x,/ operator allowed
exit;;
esac
echo Answer is $z else
echo "Usage - $0 value1 operator value2"
echo " Where, value1 and value2 are numeric values"
echo " operator can be +,-,/,x (For Multiplication)"
fi
```

While ,for loop

Shell Program to Print Numbers from 1 to 10 using While Loop

```
clear
n=1
while [ $n -le 10 ]
do
echo $n
n=`expr $n + 1`
done
```

Shell Program to Print Numbers from 1 to 10 using For Loop

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

Lab 4

1.AIM: Write a shell script to create a file in $USER /class/batch directory. Follow the Instructions.

- Input a page profile to yourself, copy it into other existing file
- Start printing file at certain line
- Print all the difference between two file, copy the two files at $USER/CSC/2007 directory.
- Print lines matching certain word pattern.

2. Software Used:

Operating System: LINUX

3. Source Code :

(i) Input a page profile to yourself, copy it into other existing file;

Solution:-

```
echo"create a file in /user/class/batch in directory"
mkdir –p user/class/b1
echo "Display present working DIR"
cd user/class/b1
pwd
echo "Enter a file name"
read file1
echo "Enter contains in $file1"
cat  > $file1
echo "Enter existing file name"
read  file2
echo "Display copy of contains $file1 to $file2"
cp  $file1   $file2
cat $file2
```

(ii) Start printing file at certain line

Solution:-

```
echo"create a file in /user/class/batch in directory"
mkdir –p user/class/b1
echo "Display present working DIR"
cd user/class/b1
pwd
```

(iii) echo "Enter a file name"

```
read file1
```

```
echo "Enter contains in $file1"
cat  > $file1
echo "Start Printing at 5 line "
tail +5 $file1
```

(iii) Print all the difference between two file, copy the two files at $USER/CSC/2007 directory.

Solution:-

```
echo "enter first file name"
read file1
echo "enter second file name"
read file2
echo "enter third file name"
read file3
echo "Enter contains to $file1"
cat> $ file1
echo "Enter contains to $file2"
cat> $ file2
echo "Display difference between $file1 and $file2  copy to $file3"
diff –a $file1 $file2 > $file3
cat $file3
```

iv)Print lines matching certain word pattern.

Solution :-

```
#mkdir IT
#cd IT
#vim assignmentno4.4.sh
echo "create a file "
read file1
echo "inputs contains in file $file1"
cat> $file1
echo "Enter word we findout "
read f
grep –ni $f $file1
```

Lab 5

1.AIM: Write shell script for-

- Showing the count of users logged in
- Printing Column list of files in your home directory.
- Listing your job with below normal priority
- Continue running your job after logging out.

2. SOFTWARE USED:
Operating System: Linux

3. SOURCE CODE:

(i) Showing the count of users logged in,
sol-> echo " Show all users login"
who
echo " count all login name"
who |wc –l

(ii)Printing Column list of files in your home directory
sol -> echo " Printing 3-colomn in a Home directory"
ls – l | cut -17-24,39 - 42,56 -

(iii)Listing your job with below normal priority
Sol-> echo "list of normal priority "
ps – al
echo – al | cut – c26-29, 70 -

(iv)Continue running your job after logging out.
nohup command-with-options &
note:- Nohup stands for no hang up

Lab 6

1. AIM: Write a shell script to change data format. Show the time taken in execution of this script.

2. SOTWARE USED:
Operating System: Linux

3. SOURCE CODE:

```
echo " Enter file name "
readfname
echo " Input contains in $fname"
cat>fname
echo "Display create file than current time "
ls –l $fname
echo " Modification $fname"
vi $ fname
echo " show access time "
ls –ult $fname
echo " show modification time "
ls –clt $ fname
```

Lab 7

1.AIM:. Write a shell script to print file names in directory showing date of creation & serial no. of file.

2. SOFTWARE USED:
Operating System: Linux

3. SOURCE CODE:

```
echo “sort login name by time”
echo “show login name”
who
echo “show only name and time “
who | gwk ‘{print $1,$5}’
echo “show sort by time “
who |gawk ‘{ print $5,$1}’
```

Output
Show only name and time
root 12:48
show sort by time
12:48 root

Lab 8

1. **AIM**: Write a shell script to count lines, words & characters in its input. (do not use wc)

2. SOTWARE USED:
Operating System: Linux

3. SOURCE CODE:

```
read –p “create file name “
fname
echo “input the contains of file “
cat> $ fname
clear
echo “ Display all record “
cat $fname
echo “ show file line , word ,char”
gawk  ‘{ nc+=length ($0)+1nw +=NF}
END ‘{print “ line =” NF , “\n word =”nw, “\n char =”nc}’ $fname
```

Output

```
Create and enter file name
Display all records
11
22
33
```

Lab 9

1. AIM: Write a shell script to print end of a Glossary file in reverse order using array.

2. SOFTWARE USED:
Operating System: Linux

3. SOURCE CODE

```
file_name="$1"

    #Check if file name is given, and it exists

if [ "$#" -eq 0 ]
then
echo "Syntax: $0 filename"
exit
elif [ ! -f "$file_name" ]
then
echo "File \""$file_name"\" does not exist"
exit
fi

    #Set the IFS variable to \n this enables reading one \n separated
    # line per read

IFS=$'\n'
declare -a arr

    #Read from file_name and store each line into next array location.
while read -r line
do
arr+=("${line}");
done< "$file_name"

    #If last line is not \n terminated read returns false, body of while
    # is not executed. Instead of saving it in the array directly print it
    # because it will be the first line in the file. Also make sure not to
    # terminate the line with newline character, so use echo -n

if [ ! -z "$line" ]
then
echo -n "$line"
fi

    #Count number of lines, and adjust index
```

```
i=${#arr[*]}
i=$((i-1))

    #Print the lines in reverse order with a newline after each line
    # (no -n after echo ensures it. Include the -E parameter to make sure
    # no slash '\' are interpreted as escape sequences
while [ $i -ge 0 ]
do
echo -E "${arr[$i]}"
  i=$((i-1))
done
```

Lab 10

1. AIM: Write a shell script to check whether Ram logged in, continue checking further after every 30 seconds till success.

2. SOTWARE USED:

Operating System: Linux

3. SOURCE CODE:

```
Invalidoptions()
{
  echo "Usage: `basename $0` [OPTIONS]"
  echo "OPTIONS:"
  echo -e "\t -d      for display today's date"
  echo -e "\t -u      for Logged in users list"
  echo -e "\t -f ARG  for Disk and Memory Statistics"
  echo -e "\t        (ARG=D for disk statistics; ARG=M for memory statistics)"
  echo -e "\t -c ARG  for Top CPU consuming process"
  echo -e "\t         (ARG=10 means top 10 process)"
  echo -e "\t -m ARG  for Top Memory consuming process"
  echo -e "\t         (ARG=10 means top 10 process)"
  echo -e "\t Note: Only one option at a time and -f,-c and -m require argument"
  exit 1
}

Isnumber()
{
  if [ $1 -eq $1 2> /dev/null ]
     then
     :
  else
     echo -e "You supplied bad argument, \"$1\" is not a number"
Invalidoptions
  fi
}

if [ $# -lt 1 -o $# -gt 2 ]
then
Invalidoptions
if [ $# -eq 1 -a "$1" != "-d" -a "$1" != "-u" -a "$1" != "-f" -a "$1" != "-c" ]
fi

then
Invalidoptions
fi

if [ $# -eq 2 ] && [ "$1" != "-f" -a "$1" != "-c" -a "$1" != "-m" ]
```

```
then
Invalidoptions
fi

choice=
top="head -$2"
while getoptsudf:c:m: choice
do
case $choice in
  d) echo -e " Today's Date: \c"
     date +"  %d-%h-%Y       Time:  %T";;
  u) echo -e "\tCurrently Logged In Users"
     who;;
  f)
    if [ "$OPTARG" = "D" ]
    then
       echo -e "\t\tDisk Statistics"
df -h | grep "%"
elif [ "$OPTARG" = "M" ]
    then
       echo -e "\t Memory Statistics "
       free -m | awk 'BEGIN{printf "\t\tTotal\tUsed\tFree\n"; OFS="\t" }\
 /Mem/||/Swap/{printf "\t"; print $1,$2,$3,$4}'
    else
Invalidoptions
    fi;;
  m) Isnumber $OPTARG
     k3sort="sort -nr -k 3"
     echo -e " PID   PPID  MEM  CPU COMMAND "
ps -Aopid= -o ppid= -o pmem= -o pcpu= -o comm=|$k3sort|$top;;
  c) Isnumber $OPTARG
     k4sort="sort -nr -k 4"
     echo -e " PID   PPID  MEM  CPU COMMAND "
ps -Aopid= -o ppid= -o pmem= -o pcpu= -o comm=|$k4sort|$top;;

esac
done
```

Output:

```
[root@localhost blog]# sh sys_monitor2.sh -u
    Currently Logged In Users
root     tty7         2009-09-23 13:48 (:0)
root     pts/2        2009-09-23 14:36 (:0.0)

[root@localhost blog]# sh sys_monitor2.sh -d
Todays Date:   23-Sep-2009       Time:  16:50:38
```

```
[root@localhost blog]# sh sys_monitor2.sh -m 5
PID   PPID  MEM  CPU COMMAND
3122  3102  9.6  3.0 firefox
2765  2540  1.9  0.0 nautilus
3849     1  1.7  1.0 ktorrent
2882     1  1.6  0.0 tomboy
2810     1  1.6  0.0 /usr/bin/sealer
```

Lab-11

1. **AIM:** Write a shell script to compute GCD & LCM of two numbers.

2. **SOTWARE USED:**
   Operating System: Linux

3. **SOURCE CODE:**

Write a shell script to compute GCD of two numbers.

```
echo Enter two numbers with space in between
read a b
m=$a
if [ $b -lt $m ]
then
m=$b
fi
while [ $m -ne 0 ]
do
x=`expr $a % $m`
y=`expr $b % $m`
if [ $x -eq 0 -a $y -eq 0 ]
then
echogcd of $a and $b is $m
break
fi
m=`expr $m - 1`
done
```

Write a shell script to compute LCM of two numbers.

```
echo "Enter first no"
read a
echo "Enter 2nd no"
read b
p= 'expr $a \* $b'
while [$b -ne 0]
do
r= 'expr $a % $b'
a=$b
b=$r
done
LCM = 'expr $p / $a'
echo "LCM = $LCM"
```

Lab 12

1. **AIM:** Write a shell script to find whether a given number is prime.

2. **SOFTWARE USED**:
   Operating System: Linux

3. **SOURCE CODE:**

```
i=2
rem=1
echo "Enter a number"
read num
if [ $num -lt 2 ]
then
echo -e "$num is not prime\n"
exit 0
fi
while [ $i -le `expr $num / 2` -a $rem -ne 0 ]
do
rem=`expr $num % $i`
i=`expr $i + 1`
done
if [ $rem -ne 0 ]
then
echo -e "$num is prime\n"
else
echo -e "$num is not prime\n"
fi
```

BEYOND CURRICULUM

EXPERIMENT NO. 1

1. AIM: Making a dictionary

2. SOURCE CODE:

```
E_BADARGS=65
if [ ! -r "$1" ] # Need at least one
then #+ valid file argument.
echo "Usage: $0 files-to-process"
exit $E_BADARGS
fi
cat $* #| Contents of specified files to stdout.

tr A-Z a-z | # Convert to lowercase.

tr ' ' '\012'| # New: change spaces to newlines.
# tr -cd '\012[a-z][0-9]' | # Get rid of everything non-alphanumeric
#+ (in original script).
tr -c '\012a-z' '\012' | # Rather than deleting non-alpha chars,
 #+ change them to newlines.
sort | # $SORT options unnecessary now.
uniq | # Remove duplicates.
grep -v '^#' | # Delete lines beginning with a hashmark.
grep -v '^$' # Delete blank lines.
exit 0
```

EXPERIMENT NO. 2

1. Shell script to perform database operations for student data like view, add and delete records in Unix / Linux.

2. SOURCE CODE:

```
clear
i="y"
echo "Enter name of database "
read db
while [ $i = "y" ]
do
clear
echo "1.View the Data Base "
echo "2.View Specific Records "
echo "3.Add Records "
echo "4.Delete Records "
echo "5.Exit "
echo "Enter your choice "
readch
case $ch in
1)cat $db;;
2)echo "Enter id "
read id
grep -i "$id" $db;;
3)echo "Enter new std id "
ireadtid
echo "Enter new name:"
readtnm
echo "Enter designation "
read des
echo "Enter college name"
read college
echo "$tid    $tnm    $des    $college">>$db;;
4)echo "Enter Id"
read id
       # set -a
   #    sed '/$id/d' $db>dbs1
grep -v "$id" $db>dbs1
echo "Record is deleted"
cat dbs1;;
5)exit;;
*)echo "Invalid choice ";;
esac
echo "Do u want to continue ?"
```

```
read i
if [ $i != "y" ]
then
exit
Fi
done
```