./

Learning Report -

Linux OS and programming

|  |  |  |
| --- | --- | --- |
| **Name** | **PS Number** | **Email ID** |
| DEEKSHA P | 99002664 | deeksha.p@ltts.com |

**Details**

# 

# 

Contents

[Contents 3](#_Toc54425711)

[Learning Objectives of the Module 5](#_Toc54425712)

[Activity 1 – Individual and Collaborative Learning 6](#_Toc54425713)

[Activity 2 – Linux Commands 7](#_Toc54425714)

[INTRO: 7](#_Toc54425715)

[OUTRO: 8](#_Toc54425716)

[BASIC COMMANDS: 9](#_Toc54425717)

[COMMANDS AND ARGUMENTS: 11](#_Toc54425718)

[CONTROL OPERANDS: 12](#_Toc54425719)

[FILE LINKS: 13](#_Toc54425720)

[STANDARD FILE PERMISSIONS: 14](#_Toc54425721)

[FILE SYSTEM: 15](#_Toc54425722)

[FILTERS: 16](#_Toc54425723)

[SHELL PATTERN MATCHING: 17](#_Toc54425724)

[SHELL HISTORY: 18](#_Toc54425725)

[SHELL VARIABLES: 19](#_Toc54425726)

[WORKING WITH DIRECTORIES: 20](#_Toc54425727)

[WORKING WITH FILES: 21](#_Toc54425728)

[WORKING WITH FILE CONTENTS: 22](#_Toc54425729)

[Activity 3 - Development Tools 23](#_Toc54425730)

[Activity 4 – Shell Script 24](#_Toc54425731)

[Activity 5 – Process, Threads and IPC 25](#_Toc54425732)

[Activity 6 – Mini Project 27](#_Toc54425733)

**List of Tables**

[Table 1: Intro 7](#_Toc53945102)

[Table 2: Outro 9](#_Toc53945103)

[Table 3: Basic Commands 10](#_Toc53945104)

[Table 4: Commands and Arguments 12](#_Toc53945105)

[Table 5: Control Operands 12](#_Toc53945106)

[Table 6: File Links 13](#_Toc53945107)

[Table 7: Standard File Permissions 14](#_Toc53945108)

[Table 8: File System 15](#_Toc53945109)

[Table 9: Filters 16](#_Toc53945110)

[Table 10: Shell Pattern matching 17](#_Toc53945111)

[Table 11: Shell History 18](#_Toc53945112)

[Table 12: Shell variables 19](#_Toc53945113)

[Table 13: Working with Directories 20](#_Toc53945114)

[Table 14: Working with Files 21](#_Toc53945115)

[Table 15: Working with File contents 22](#_Toc53945116)

# Learning Objectives of the Module

* Knowledge on features of shell and commands to handle files and directories in Linux
* Apply the knowledge of GNU tools to build, debug and analyze C/C++ programs
* Apply and analyze concepts of Process management and IPC to develop system programs
* Apply the knowledge of commands to analyze, signal the running processes from CLI.

**Capture Brief on the below aspects for each activity:**

* Major Stages of Learning while going through the activity
* Challenges Faced and how were they overcome.
* Good Learning resources.
* Observations and learning from Peer review for each activity

Note:

* Provide links to SharePoint folder and public repository wherever applicable
* Include progress related links on online courses if any.
* Other links to the work done related to the module.

# Activity 1 – Individual and Collaborative Learning

Linux was a total new concept and approach for me. The learning sources provided by faculty and my peers helped me a lot to travel in the transition path from Windows to Linux. Transition was quite difficult, as I was a continuous user of windows.

**Key Learning:**

* GNU tools for compilation and execution of the C/C++ programs.
* Usage of shell and commands and able to handle files and directories.
* Usage of command line for most of the tasks.
* Understanding Shell scripting.
* Understanding Linux programming concepts like process, threads, etc.,
* Understanding IPC techniques.

**Challenges and Overcoming of it:**

* Using Linux was tedious initially but overcame it by practicing command line interface.
* Understanding of different concepts in OS like Semaphores was quite difficult. Worked on concepts by using learning sources provided.

**Good Learning Resources:**

Learning resources provided by faculty and my peers helped a lot, as it’s difficult to choose from a huge bunch of sources that's available online.

* Learnt different commands on the terminal.
* Learnt to download and install different applications through the terminal.
* Used visual studio.
* Being new to the LINUX OS it was difficult to grasp at a very high speed.

# Activity 2 – Linux Commands

 Few basics commands include:

|  |  |
| --- | --- |
| **Commands** | **Description** |
| **Pwd** | It gives us the absolute path, which means the path that starts from the root. |
| **ls** | Use the "ls" command to know what files are in the directory you are in. You can see all the hidden files by using the command “ls -a”. |
| **cd** | Use the "cd" command to go to a directory. |
| **mkdir & rmdir** | Use the mkdir command when you need to create a folder or a directory. |
| **touch** | The touch command is used to create a file. |
| **man & --help** | To know more about a command and how to use it, use the man command. |
| **cp and mv** | Use the cp command to copy files through the command line. Use the mv command to move files through the command line. |

* Various linux commands execution was demonstrated during the session along with hands-on. Few of the basic commands are:
* Date, cal, echo, bc, uname, clear, seq, rev, wc, cat, tac, date, ls
* Commands related to file locations, file type, file permissions and disk management. Few of them are: Locate, type, chamod, chown, chgrp, df, du, Mount, unmount/

## INTRO:

|  |  |  |
| --- | --- | --- |
| 1 | Who initiated Linux Kernel Development? | Linus Torvalds |
| 2 | Who initiated GNU Software Development? | Richard Stallman |
| 3 | What is the full form of GNU? | GNU's Not Unix |
| 4 | What is GNU/Linux? | Linux is the kernel. Linux is normally used in combination with the GNU operating system: the whole system is basically GNU with Linux added, or GNU/Linux. |
| 5 | Do Linux is Free and Open Source software? | Yes |
| 6 | What are the various Open Source Licensing Models, e.g. GPL v3? | GNU General Public License (GPL)  The Apache License.  Microsoft Public Licenses (Ms-PL)  Berkeley Software Distribution (BSD)  Common Development and Distribution License (CDDL)  Eclipse Public License (EPL)  MIT License. |
| 7 | What is the full form of GPL? | GNU General Public License |
| 8 | Identify some popular Linux Distributions | Ubuntu Linux  Red Hat Enterprise Linux  Linux Mint  Debian |
| 9 | Fedora Identify some popular Linux Desktop Environments | KDE  MATE  GNOME  Cinnamon  Budgie  LXQt  Xfce  Deepin |
| 10 | Identify some end user applications in Linux (Alternatives to Windows apps) | PDF Editor â€“ LibreOffice Draw  LaTeX Editor â€“ TeXmaker  Web Browser â€“ Chromium  Free Office Suite â€“ LibreOffice  Image Editor â€“ GIMP  Linux PDF Viewer â€“ Okular |
| 11 | Identify the Programming support in Linux | PHP, Perl, Ruby, Python, Java, Go, Rust, Haskell |

Table 1: Intro

## OUTRO:

|  |  |  |
| --- | --- | --- |
| 1 | What is a shell? | A shell is a program that receives commands from the user and gives it to the OS to process, and it shows the output. |
| 2 | What is a terminal? | A shell is a program that receives commands from the user and gives it to the OS to process, and it shows the output. |
| 3 | What is a shell prompt? | The shell prompt (or command line) is where one types commands. |
| 4 | Identify default Primary, Secondary prompt in the shell? | The default command prompt is a single character (typically $ or #). When you issue a command that is incomplete, the shell will display a secondary prompt and wait for you to complete the command and hit Enter again. The default secondary prompt is > (the greater than sign) |
| 5 | What is the typical syntax of a command? | Command [options] [arguments] |
| 6 | How to specify single letter Vs multi lettered options with commands | Single letter options start with -  Multi lettered options start with -- |
| 7 | List out some available shells for Linux | bash, ksh, csh, tcsh, zsh |
| 8 | What is the default shell in your machine? | Bash |
| 9 | What is the full form of Bash | Bourne Again Shell |
| 10 | What is the difference between pseudo terminal and login console? | A pseudo terminal (PTY) is a user level program that appears to be a terminal device to another program. The console is the text output device for system administration messages. These messages come from the kernel, from the init system and from the system logger. |
| 11 | How can you navigate between recent commands? | Use the Up arrow key |
| 12 | How can you auto complete command, file/directory names? | Use the Tab key |
| 13 | What is the significance of following key strokes | Ctrl + C, Ctrl + \, Ctrl + Z, Ctrl + U, Ctrl + D Ctrl + L?  Ctrl+C - Interrupt (kill) the current foreground process  Ctrl + \- Quit process (by SIGQUIT signal)  Ctrl + Z- Suspend process (by SIGTSTP signal)  Ctrl + U- Discard the current command  Ctrl + D- End of input  Ctrl + L- Clear the screen |
| 14 | How to refer documentation of commands? | man |
| 15 | What are the sections of man pages? | Executable programs or shell commands  System calls (functions provided by the kernel)  Library calls (functions within program libraries)  Special files (usually found in /dev)  File formats and conventions eg /etc/passwd  Games  Miscellaneous  System administration commands (usually only for root)  Kernel routines |
| 16 | How to quit man page view? | press q |

Table 2: Outro

## 

## BASIC COMMANDS:

|  |  |  |
| --- | --- | --- |
| 1 | Explain the difference between these two commands.   * find /data -name "\*.txt" * find /data -name \*.txt | * When the \*.txt is quoted then the shell will not touch it. The find tool will look in the /data for all files ending in .txt. * When \*.txt is not quoted then the shell might expand this “when one or more files that ends in .txt exist in the current directory”. The find might show a different result, or can result in a syntax error. |
| 2 | Explain the difference between these two statements. Will they both work when there are 200 .odf files in /data? How about when there are 2 million .odf files?   * find /data -name "\*.odf" > data\_odf.txt * find /data/\*.odf > data\_odf.txt | * The first find will output all .odf filenames in /data and all subdirectories. The shell will redirect this to a file. * The second find will output all files named .odf in /data and will also output all files that exist in directories named \*.odf (in /data). * With the two millions of files the command line would be expanded beyond the maximum that the shell can accept. Then the last part of the command line would be lost. |
| 3 | Write a find command that finds all files created after January 30th 2010. | touch -t 201001302359 marker\_date  find . -type f -newer marker\_date |
| 4 | Write a find command that finds all \*.odf files created in September 2009. | touch -t 200908312359 marker\_start  touch -t 200910010000 marker\_end  find . -type f -name "\*.odf" -newer marker\_start ! -newer marker\_end |
| 5 | Count the number of \*.conf files in /etc and all its subdirs. | find /etc -type f -name '\*.conf' | wc –l |
| 6 | Here are two commands that do the same thing: copy \*.odf files to /backup/ . What would be a reason to replace the first command with the second?   * cp -r /data/\*.odf /backup/ * find /data -name "\*.odf" -exec cp {} /backup/ \; | The first might fail when there are too many files to fit on one command line |
| 7 | Create a file called loctest.txt. Can you find this file with locate? Why not? How do you make locate find this file? | You cannot locate this with locate because it is not yet in the index.  By using “updatedb” we can make locate find this file. |
| 8 | Use find and -exec to rename all .htm files to .html. | find . -name '\*.htm'\  find . -name '\*.htm' -exec mv {} {}l \;  find . -name '\*.htm\*' |
| 9 | Issue the date command. Now display the date in YYYY/MM/DD format. | date +%Y/%m/%d |
| 10 | Issue the cal command. Display a calendar of 1582 and 1752. Notice anything special ? | * cal * cal -y 1582 * displays 1582 calendar:   In 1582, when Pope Gregory XIII introduced his Gregorian calendar, Europe adhered to the Julian calendar, first implemented by Julius Caesar in 46 B.C. Since the Roman emperor's system miscalculated the length of the solar year by 11 minutes, the calendar had since fallen out of sync with the seasons.   * cal -y 1752 * displays 1752 calendar:   Since the Gregorian calendar accounted more accurately for leap years, it was 11 days ahead of the Julian calendar by 1752. |

Table 3: Basic Commands

## COMMANDS AND ARGUMENTS:

|  |  |  |
| --- | --- | --- |
| 1 | How many arguments are in this line (not counting the command itself).   * touch '/etc/cron/cron.allow' 'file 42.txt' "file 33.txt | Three (3) |
| 2 | Is tac a shell builtin command? | Tac command in linux is used to concatenate and print files in reverse. This command will write each file to standard output, the last line first. When no file is specified then this command will read the standard input.  • Tac is /usr/bin/tac |
| 3 | Is there an existing alias for rm? | * There is no inbuit rm in alias * bash:alias:rm:not found * if we need we can create using “alias rm” |
| 4 | What is -i option of rm. Create and remove a file to test the -i option. | * ‘-I’ option in rm command will prompt before deleting a file. * man rm   touch testfile  rm –i testfile |
| 5 | Execute: alias rm='rm -i'. Test your alias with a test file. Does this work as expected? | * When we need to be promted ,just use rm -I in the alias. * touch testfile rm testfile |
| 6 | List all current aliases. | * For this we need to type alias at the promt and any active aliases will be listed. * Alias |
| 7 | 1. Create an alias called 'city' that echoes your hometown. 2. Use your alias to test that it works. | * alias city= ‘echo bvrm’ * alias * alias city=’echo bvrm’ * alias egrep=’egrep –color=auto’ * alias fgrep=’egrep –color=auto’ * alias grep=’egrep –color=auto’ * alias l=’ls -CF’ * alias la=’ls -A’ * alias ll=’ls –alF’ * alias ls=’ls –color=auto’ |
| 8 | Execute set -x to display shell expansion for every command. | * city (it should display bvrm) bvrm |
| 9 | Test the functionality of set -x by executing your city and rm aliases. | * set –x |
| 10 | Execute set +x to stop displaying shell expansion. | * ~$ set -x * ~$ city * + echo bvrm * bvrm |
| 11 | Remove your city alias. | * unalias city * unalias:removes the city name bvrm |
| 12 | What is the location of the cat and the passwd commands ? | * cat: /bin/cat * passwd: /user/bin/passwd |
| 13 | Explain the difference between the following commands:  echo  /bin/echo | The echo command will be interpreted by the shell as the built-in echo command. The /bin/echo command will make the shell execute the echo binary located in the /bin director |
| 14 | Explain the difference between the following commands:  echo Hello  echo -n Hello | * The -n option of the echo command will prevent echo from echoing a trailing newline. * echo Hello will echo six characters in total, echo -n hello only echoes five characters. |
| 15 | Display A B C with two spaces between B and C. | * echo "A B C" |
| 16 | Display (do not use spaces) exactly the following output:  4+4 =8  10+14 =24 | * echo -e "4+4\t=8" ; * echo -e "10+14\t=24" |
| 17 | Use echo to display the following exactly :??\\  Find two solutions with single quotes, two with double quotes and one without quotes. | * echo '??\\' * echo -e '??\\\\' * echo "??\\\\" * echo -e "??\\\\\\" * echo ??\\\\ |
| 18 | Use one echo command to display three words on three lines. | * echo -e “firstword \nsecondword \nthirdword" “\n” ->takes to next line |

Table 4: Commands and Arguments

## CONTROL OPERANDS:

|  |  |  |
| --- | --- | --- |
| 1 | When you type passwd, which file is executed? | which passwd |
| 2 | What kind of file is that? | file /usr/bin/passwd |
| 3 | Execute the pwd command twice. (remember 0.) | pwd; pwd |
| 4 | Execute ls after cd /etc, but only if cd /etc did not error. | cd /etc && ls |
| 5 | Execute cd /etc after cd etc, but only if cd etc fails. | cd /etc || cd /etc |
| 6 | Echo it worked when 'touch test42' works, and echo it failed when the touch failed. All on one command line as a normal user (not root). Test this line in your home directory and in /bin/ . | * ~$ cd ; touch test42 && echo it worked || echo it failed   It worked   * ~$ cd /bin; touch test42 && echo it worked || echo it failed   touch: cannot touch `test42` : Permission denied  it failed |
| 7 | Execute sleep 6, what is this command doing ? | pausing for six seconds |
| 8 | Execute sleep 200 in background (do not wait for it to finish). | sleep 200 & |
| 9 | Write a command line that executes rm file55. Your command line should print 'success' if file55 is removed, and print 'failed' if there was a problem. | * rm file55 && echo success || echo failed |
| 10 | Use echo to display "Hello World with strange' characters \ \* [ } ~ \\ ." (including all quotes) | * echo \”Hello World with strange\’ characters \\ \\* \[ \} \~ \\\\ \. \”   or   * echo \”Hello World with strange\’ characters \ \* [ } ~ \\ . “\” |

Table 5: Control Operands

## FILE LINKS:

|  |  |  |
| --- | --- | --- |
| 1 | Create two files named winter.txt and summer.txt, put some text in them. | * echo cold > winter.txt ; echo hot > summer.txt |
| 2 | Create a hard link to winter.txt named hlwinter.txt. | * ln winter.txt hlwinter.txt |
| 3 | Display the inode numbers of these three files, does the hard links have the same inode? | * ls -li winter.txt summer.txt hlwinter.txt |
| 4 | Use the find command to list the two hardlinked files | * find . inum xyz #replace xyz with the inode number |
| 5 | Everything about a file is in the inode, except two things : name them! | * The name of the file is in a directory,and the contents is somewhere on the disk |
| 6 | Create a symbolic link to summer.txt called slsummer.txt. | * ln -s summer.txt slsummer.txt |
| 7 | Find all files with inode number 2. What does this information tell you ? | * It tells you there is more than one inode table (one for every formatted partition + virtual file systems) |
| 8 | Look at the directories /etc/init.d/ /etc/rc2.d/ /etc/rc3.d/ ... do you see the links ? | * ls -l /etc/init.d * ls -l /etc/rc2.d * ls -l /etc/rc3.d |
| 9 | Look in /lib with ls -l... | * ls -l /lib |
| 10 | Use find to look in your home directory for regular files that do not(!) have one hard link. | * find ~ ! -links 1 -type f |

Table 6: File Links

## STANDARD FILE PERMISSIONS:

|  |  |  |
| --- | --- | --- |
| 1 | As normal user, create a directory ~/permissions. Create a file owned by yourself in there. | * mkdir ~/permissions ; touch ~/permissions/myfile.txt |
| 2 | Copy a file owned by root from /etc/ to your permissions dir, who owns this file now ? | * cp */*etc*/*hosts ~/permissions/   The copy is owned by the user |
| 3 | As root, create a file in the users ~/permissions directory. | * (become root)# touch /home/username/permissions/rootfile |
| 4 | As normal user, look at who owns this file created by root. | * ls -l ~/permissions   The file is created by root is owned by root |
| 5 | Change the ownership of all files in ~/permissions to yourself. | * chown user ~/permissions/\*   Cannot become owner of the file that belongs to root |
| 6 | Make sure you have all rights to these files, and others can only read. | * chmod 644 (on file) * chmod 755 (on directories) |
| 7 | With chmod, is 770 the same as rwxrwx--- ? | yes |
| 8 | With chmod, is 664 the same as r-xr-xr-- ? | no |
| 9 | With chmod, is 400 the same as r-------- ? | yes |
| 10 | With chmod, is 734 the same as rwxr-xr-- ? | no |
| 11 | a. Display the umask in octal and in symbolic form.  b. Set the umask to 077, but use the symbolic format to set it. Verify that this works. | a. umask ; umask -S  b. umask -S u=rwx, g0= |
| 12 | Create a file as root, give only read to others. Can a normal user read this file ? Test writing to this file with nano. | * (become root) * # echo hello > /home/username/root.txt * # chmod 744 > /home/username/root.txt * (become user) * vi ~/root.txt |
| 13 | a. Create a file as normal user, give only read to others. Can another normal user read this file ? Test writing to this file with vi.  b. Can root read this file ? Can root write to this file with vi ? | a. echo hello > file ; chmod 744 file  Yes, others can read this file  b. Yes, root can read and write to this file. Permissions do not apply to root |
| 14 | Create a directory that belongs to a group, where every member of that group can read and write to files, and create files. Make sure that people can only delete their own files. | * mkdir /home/project42 ; groupadd project42 * chgrp project42 /home/project42 ; chmod 755 /home/project42 |

Table 7: Standard File Permissions

## FILE SYSTEM:

|  |  |  |
| --- | --- | --- |
| 1 | Put a sorted list of all bash users in bashusers.txt. | * grep bash /etc/passwd | cut -d: -f1 | sort > bashusers.txt |
| 2 | Put a sorted list of all logged on users in onlineusers.txt. | * who | cut -d' ' -f1 | sort > onlineusers.txt |
| 3 | Make a list of all filenames in /etc that contain the string conf in their filename. | * ls /etc | grep conf |
| 4 | Make a sorted list of all files in /etc that contain the case insensitive string conf in their filename. | * ls /etc | grep -i conf | sort |
| 5 | Look at the output of /sbin/ifconfig. Write a line that displays only ip address and the subnet mask. | * /sbin/ifconfig | head -2 | grep 'inet ' | tr -s ' ' | cut -d' ' -f3,5 |
| 6 | Write a line that removes all non-letters from a stream. | * ~$ cat text   This is, yes really! , a text with ?&\* too many str$ange# characters ;-)   * ~$ cat text | tr -d ',!$?.\*&^%#@;()-'   This is yes really a text with too many strange characters |
| 7 | Write a line that receives a text file, and outputs all words on a separate line. | * ~$ cat text2   it is very cold today without the sun   * ~$ cat text2 | tr ' ' '\n'   it  is  very  cold  today  without  the  sun |
| 8 | Write a spell checker on the command line. (There may be a dictionary in /usr/share/dict/ .) | * ~$ echo "The zun is shining today" > text * ~$ cat > DICT   is  shining  sun  the  today   * ~$ cat text | tr 'A-Z ' 'a-z\n' | sort | uniq | comm -23 - DICT   zun |

Table 8: File System

## FILTERS:

|  |  |  |
| --- | --- | --- |
| 1 | Put a sorted list of all bash users in bashusers.txt. | * grep bash /etc/passwd | cut -d: -f1 | sort > bashusers.txt |
| 2 | Put a sorted list of all logged on users in onlineusers.txt. | * who | cut -d' ' -f1 | sort > onlineusers.txt |
| 3 | Make a list of all filenames in /etc that contain the string conf in their filename. | * ls /etc | grep conf |
| 4 | Make a sorted list of all files in /etc that contain the case insensitive string conf in their filename. | * ls /etc | grep -i conf | sort |
| 5 | Look at the output of /sbin/ifconfig. Write a line that displays only ip address and the subnet mask. | * /sbin/ifconfig | head -2 | grep 'inet ' | tr -s ' ' | cut -d' ' -f3,5 |
| 6 | Write a line that removes all non-letters from a stream. | * ~$ cat text   This is, yes really! , a text with ?&\* too many str$ange# characters ;-)   * ~$ cat text | tr -d ',!$?.\*&^%#@;()-'   This is yes really a text with too many strange characters |
| 7 | Write a line that receives a text file, and outputs all words on a separate line. | * ~$ cat text2   it is very cold today without the sun   * ~$ cat text2 | tr ' ' '\n'   it  is  very  cold  today  without  the  sun |
| 8 | Write a spell checker on the command line. (There may be a dictionary in /usr/share/dict/ .) | * ~$ echo "The zun is shining today" > text * ~$ cat > DICT   is  shining  sun  the  today   * ~$ cat text | tr 'A-Z ' 'a-z\n' | sort | uniq | comm -23 - DICT   zun |

Table 9: Filters

## SHELL PATTERN MATCHING:

|  |  |  |
| --- | --- | --- |
| 1 | Create a test directory and enter it. | * mkdir testdir; cd testdir |
| 2 | Create the following files :  file1  file10  file11  file2  File2  File3  file33  fileAB  filea  fileA  fileAAA  file(  file 2  (the last one has 6 characters including a space) | * touch file1 file10 file11 file2 File2 File3 * touch file33 fileAB file fileA fileAAA * touch "file(" * touch "file 2" |
| 3 | List (with ls) all files starting with file | * ls file\* |
| 4 | List (with ls) all files starting with File | * ls File\* |
| 5 | List (with ls) all files starting with file and ending in a number. | * ls file\*[0-9] |
| 6 | List (with ls) all files starting with file and ending with a letter | * ls file\*[a-z] |
| 7 | List (with ls) all files starting with File and having a digit as fifth character. | * ls File[0-9]\* |
| 8 | List (with ls) all files starting with File and having a digit as fifth character and nothing else. | * ls File[0-9] |
| 9 | List (with ls) all files starting with a letter and ending in a number. | * ls [a-z]\*[0-9] |
| 10 | List (with ls) all files that have exactly five characters. | * ls ????? |
| 11 | List (with ls) all files that start with f or F and end with 3 or A. | * ls [fF]\*[3A] |
| 12 | List (with ls) all files that start with f have i or R as second character and end in a number. | * ls f[iR]\*[0-9] |
| 13 | List all files that do not start with the letter F. | * ls [!F]\* |
| 14 | Can echo replace ls ? How can you list the files in the current directory with echo ? | * echo \* |
| 15 | Is there another command besides cd to change directories ? | * pushd popd |

Table 10: Shell Pattern matching

## 

## SHELL HISTORY:

|  |  |  |
| --- | --- | --- |
| 1 | Issue the command  echo | * echo   The answer to the meaning of life, the universe and everything is 42 |
| 2 | Repeat the previous command using only two characters (there are two solutions!) | !!  OR  !e |
| 3 | Display the last 5 commands you typed. | * ~$ history 5 * 52 ls -l * 53 ls * 54 df -h | grep sda * 55 echo The answer to the meaning of life, the universe and everything is 42 * 56 history 5 |
| 4 | Issue the long echo from question 1 again, using the line numbers you received from the command in question 3. | * ~$ !55   echo The answer to the meaning of life, the universe and everything is 42  The answer to the meaning of life, the universe and everything is 42 |
| 5 | How many commands can be kept in memory for your current shell session? | * echo $HISTSIZE |
| 6 | Is the current session history stored to ~/.bashrc\_history? | Yes |
| 7 | Where are these commands stored when exiting the shell? | * echo $HISTFILE |
| 8 | How many commands can be written to the history file when exiting your current shell session? | * echo $HISTFILESIZE |
| 9 | Make sure your current bash shell remembers the only 10 commands you type. | * HISTSIZE=10 |
| 10 | When is command history written to the history file? | * For each terminal execution. |

Table 11: Shell History

## 

## SHELL VARIABLES:

|  |  |  |
| --- | --- | --- |
| 1 | Use echo to display Hello followed by your username. (use a bash variable!) | * echo Hello $USER |
| 2 | Create a variable answer with a value of 42. | answer=42 |
| 3 | Copy the value of $LANG to $MyLANG. | * MyLANG=$LANG |
| 4 | List all current shell variables. | * set * set|more on Ubuntu/Debian |
| 5 | List all exported shell variables. | * env * export * declare -x |
| 6 | Do the env and set commands display your variable ? | * env | more * set | more |
| 7 | Destroy your answer variable. | unset answer |
| 8 | Create two variables, and export one of them. | * var1=1; export var2=2 |
| 9 | Display the exported variable in an interactive child shell. | * bash * echo $var2 |
| 10 | Create a variable, give it the value 'Dumb', create another variable with value 'do'. Use echo and the two variables to echo Dumbledore. | * varx=Dumb; vary=do * echo ${varx}le${vary}re * solution by Yves from Dexia : echo $varx'le'$vary're' * solution by Erwin from Telenet : echo "$varx"le"$vary"re |
| 11 | Find the list of backslash escaped characters in the manual of bash. Add the time to your PS1 prompt. | * PS1='\t \u@\h \W$ ' |

Table 12: Shell variables

## WORKING WITH DIRECTORIES:

|  |  |  |
| --- | --- | --- |
| 1 | Display your current directory. | * pwd |
| 2 | Change to the /etc directory. | * cd /etc |
| 3 | Now change to your home directory using only three key presses. | * cd (and the enter key) |
| 4 | Change to the /boot/grub directory using only eleven key presses. | * cd /boot/grub (use the tab key) |
| 5 | Go to the parent directory of the current directory. | * cd .. (with space between cd and ..) |
| 6 | Go to the root directory. | * cd / |
| 7 | List the contents of the root directory. | * ls |
| 8 | List a long listing of the root directory. | * ls -l |
| 9 | Stay where you are, and list the contents of /etc. | * ls /etc |
| 10 | Stay where you are, and list the contents of /bin and /sbin. | * ls /bin /sbin |
| 11 | Stay where you are, and list the contents of ~. | * ls ~ |
| 12 | List all the files (including hidden files) in your home directory. | * ls -al ~ |
| 13 | List the files in /boot in a human readable format. | * ls -lh /boot |
| 14 | Create a directory testdir in your home directory. | * mkdir ~/testdir |
| 15 | Change to the /etc directory, stay here and create a directory newdir in your home directory. | * cd /etc ; mkdir ~/newdir |
| 16 | Create in one command the directories ~/dir1/dir2/dir3 (dir3 is a subdirectory from dir2,and dir2 is a subdirectory from dir1 ). | * mkdir -p ~/dir1/dir2/dir3 |
| 17 | Remove the directory testdir. | * rmdir testdir |
| 18 | Open manual page for bash and read about pushd and popd by searching in manpage. | * man bash # opens the manual * /pushd # searches for pushd * n # next (do this two/three times) |

Table 13: Working with Directories

## 

## WORKING WITH FILES:

|  |  |  |
| --- | --- | --- |
| 1 | List the files in the /bin directory | * ls /bin |
| 2 | Display the type of file of /bin/cat, /etc/passwd and /usr/bin/passwd. | * file /bin/cat /etc/passwd /usr/bin/passwd |
| 3 | Use files LFS.png dummy.pdf  a. Display the type of file of wolf.jpg and dummy.pdf  b. Rename LFS.png to wolf.pdf  c. Display the type of file of wolf.pdf and dummy.pdf. | a. file wolf.jpg dummy.pdf  b. mv LFS.png wolf.pdf  c. file wolf.pdf dummy.pdf |
| 4 | Create a directory ~/touched and enter it. | * mkdir ~/touched ; cd ~/touched |
| 5 | Create the files today.txt and yesterday.txt in touched. | * touch today.txt yesterday.txt |
| 6 | Change the date on yesterday.txt to match yesterday's date. | * touch -t 200810251405 yesterday.txt (substitute 20081025 with yesterday) |
| 7 | Copy yesterday.txt to copy.yesterday.txt | * cp yesterday.txt copy.yesterday.txt |
| 8 | Rename copy.yesterday.txt to kim | * mv copy.yesterday.txt kim |
| 9 | Create a directory called ~/testbackup and copy all files from ~/touched into it. | * mkdir ~/testbackup ; cp -r ~/touched ~/testbackup/ |
| 10 | Use one command to remove the directory ~/testbackup and all files in it. | * rm -rf ~/testbackup |
| 11 | Create a directory ~/etcbackup and copy all \*.conf files from /etc into it. Did you include all subdirectories of /etc ? | * cp -r /etc/\*.conf ~/etcbackup * Only \*.conf files that are directly in /etc/ are copied. |
| 12 | Use rename to rename all \*.conf files to \*.backup . | * On RHEL: touch 1.conf 2.conf ; rename conf backup \*.conf * On Debian: touch 1.conf 2.conf ; rename 's/conf/backup/' \*.conf |

Table 14: Working with Files

## 

## WORKING WITH FILE CONTENTS:

|  |  |  |
| --- | --- | --- |
| 1 | Display the first 12 lines of /etc/services. | * head -12 /etc/services |
| 2 | Display the last line of /etc/passwd. | * tail -1 /etc/passwd |
| 3 | Use cat to create a file named count.txt that looks like this:  One  Two  Three  Four  Five | * cat > count.txt   One  Two  Three  Four  Five (followed by Ctrl-d) |
| 4 | Use cp to make a backup of this file to cnt.txt. | * cp count.txt cnt.txt |
| 5 | Use cat to make a backup of this file to catcnt.txt. | * cat count.txt > catcnt.txt |
| 6 | Display catcnt.txt, but with all lines in reverse order (the last line first). | * tac catcnt.txt |
| 7 | Use more to display /etc/services. | * more /etc/services |
| 8 | Display the readable character strings from the /usr/bin/passwd command. | * strings /usr/bin/passwd |
| 9 | Use ls to find the biggest file in /etc. | * ls -lrS /etc |
| 10 | Use cat to create a file named tailing.txt that contains the contents of tailing.txt followed by the contents of /etc/passwd. | * cat /etc/passwd >> tailing.txt |
| 11 | Use cat to create a file named tailing.txt that contains the contents of tailing.txt preceded by the contents of /etc/passwd. | * mv tailing.txt tmp.txt ; cat /etc/passwd tmp.txt > tailing.txt |

**Table 15: Working with File contents**

# Activity 3 - Development Tools

* **Step-0 : Preparation**

Develop the following functions in respective source files, in a subdir called `src`

* mystrlen, mystrcpy, mystrcat, mystrcmp
* factorial, isPrime, isPalindrome, vsum
* set, reset, flip, query
* Creating 3 source .c files for String, Utils and BitMasking
* Provide prototypes in different header files, in a subdir called `include`
* mystring.h, myutils.h, bitmask.h
* Write a main code to invoke above functions, say `main.c`

Link: <https://github.com/99002664/Linux_Activity/tree/main/src>

* **Step-1A : Simple Makefile**
* Write a simple Makefile, assuming all files are in same location

Link: <https://github.com/99002664/Linux_Activity/blob/main/src/Makefile>

* **Step-1B : Simple Makefile with subdir `src` and header files are in subdir `include`**
* Develop the Makefile, to compile all source files and link with main.c
* Ensure that source files are in subdir `src` and header files are in subdir `include`
* Optionally you may further classify files under `src` into three more subdirs.

Link: <https://github.com/99002664/Linux_Activity/tree/main/Makefile_1B>

* **Step-2 : Static Libraries**
* Generate 3 static libraries
* libmystr.a with string functions
* libmyutils.a with utility functions
* libmasking.a with bit introspection functions
* Link the static libraries with test code
* Test the statically linked executable
* Analyse all intermediate, final outcome as per the discussion in session

Link: <https://github.com/99002664/Linux_Activity/tree/main/Static_libraries>

* **Step-3 : Dynamic Libraries**
* Generate 3 shared object files
* libmystr.so with string functions
* libmyutils.so with utility functions
* libmasking.so with bit introspection functions
* Link the shared libraries with test code
* Test the dynamically linked executable
* Analyse all intermediate, final outcome as per the discussion in session

Link: <https://github.com/99002664/Linux_Activity/tree/main/Dynamic_libraries>

**Git Repository Link:** <https://github.com/99002664/Linux_Activity>

# Activity 4 – Shell Script

Shell programming can be accomplished by directly executing shell commands at the shell prompt or by storing them in the order of execution, in a text file, called a shell script, and then executing the shell script.

**Micro project - Shell Script assignment**

**Problem Statement:**

* Script should clone a repository (read from Input.CSV file) which has program files and a Makefile.
* Navigate to repo directory and use ‘make’ to build the code. Build Success/Failure to the Results.CSV file.
* Run Cppcheck, log the number of errors/warning reported to a separate column in Results.CSV file.
* Run Valgrind, log the number of errors/warning Summary reported to a separate column in Results.CSV file.
* Run steps 2-4 for each entry in Input.CSV file.
* Display all the Results with the Input.CSV file details onto Results.CSV file.

Link: <https://github.com/99002664/LINUX_MicroProject>

# Activity 5 – Process, Threads and IPC

**Process:**

The process is the OS’s abstraction for execution

* the unit of execution
* a unit of scheduling
* the dynamic execution context

Process is often called a job, task, or sequential process. A process is identified by its Process ID (PID). In the OS, processes are represented by entries in a Process Table (PT). PID “points to” a PT entry PT entry = Process Control Block (PCB). PCB is a large data structure that contains or points to all info about the process.

On a Linux system, a process is created using the fork system call. A previously running process executes the fork system call and the result is a clone of the process that executed fork. So, we have two processes, a parent process that executed the fork system call and a newly born child process that is (almost) a copy of the parent. Create a new (child) process – fork(); Allocates new PCB. Clones the calling process (almost) Copy of parent process address space Copies resources in kernel (e.g. files). Places PCB on Ready queue. Return from fork() call 0 for child child PID for parent

**Threads:**

A thread is the execution of a program or procedure within the context of a Unix or Windows process i.e., a specialization of the concept of process.

A thread has its own- Program counter, registers, Stack.

A thread shares- Address space, heap, static data.

All other resources with other threads in the same process

* By default, a process has a single thread of execution. It is possible to have multiple threads of execution in a process. Or, in other words, we can have multiple threads in a process. Each thread has a starting function in the process's code segment. All threads share the process's code and global data.

From POSIX pthreads API:

1. int **pthread\_create** (pthread\_t \*thread, const pthread\_attr\_t \*attr, void\*(\*start\_routine) (void), void \*arg) ; creates a new thread of control. New thread begins executing at start\_routine
2. **pthread\_exit** (void \*value\_ptr); terminates the calling thread
3. **pthread\_join** (pthread\_t thread, void \*\*value\_ptr) ; blocks the calling thread until the thread specified terminates
4. pthread \_t **pthread\_self**() ; Returns the calling thread's identifier

**IPCs (INTER PROCESS COMMUNICATION)**

Linux supports three types of interprocess communication mechanisms that first appeared in Unix TM System V (1983). These are message queues, semaphores and shared memory.

The Linux IPC (Inter-process communication) facilities provide a method for multiple processes to communicate with one another. There are several methods of IPC available to Linux C programmers:

* Half-duplex UNIX pipes
* FIFOs (named pipes)
* SYSV style message queues
* SYSV style semaphore sets
* SYSV style shared memory segments
* Networking sockets (Berkeley style) (not covered in this paper)
* Full-duplex pipes (STREAMS pipes) (not covered in this paper)

These facilities, when used effectively, provide a solid framework for client/server development on any UNIX system (including Linux).

**Key Learning’s:**

* Understanding the concepts related to Semaphores, Mutex, process, threads, mutual exclusion, race condition, Message queues, FIFO/Pipes and Deadlock.

**Assignments:**

* **Assignment\_Threads**

Link: <https://github.com/99002664/LINUX_Assignments/tree/main/Assignment_Threads>

* **Assignment\_Semaphores**

Link: <https://github.com/99002664/LINUX_Assignments/tree/main/Assignment_Semaphores>

# Activity 6 – Mini Project

**Title:**

**Restaurant Table Management System**

**Problem Statement:**

The customer books a table in a Restaurant; the Manager gives the availability of that table. If the restaurant is full then customer has to wait in the lobby. This is a producer consumer problem. The Table is effectively managed by this technique in a Restaurant.

**Link:**

<https://github.com/99002664/Linux_Miniproject>

# 