Lab Experiments

1 Experiment 1: Introduction to Linux Commands

1.1 Objective

The objective of this lab experiment is to introduce students to fundamental Linux commands used for navigating the file system, managing files and directories, and performing basic system operations. By the end of this experiment, students should be familiar with commonly used commands such as ls, cd, mkdir, rm, cp, and mv, gaining a foundational understanding of the Linux command-line interface.

1.2 Reference Material[1][3][2]

1.2.1 Important Linux Commands for File and Directory Management

Command	Description	Example
`ls`	List directory contents	`ls -l`
`cd`	Change the current directory	`cd Documents`
`pwd`	Print the current working directory	`pwd`
`mkdir`	Make directories	`mkdir new_directory`
`rmdir`	Remove directories	`rmdir directory_to_remove`
`cp`	Copy files and directories	`cp file1.txt file2.txt`
`mv`	Move or rename files and directories	`mv file1.txt new_location`
`rm`	Remove files or directories	`rm file_to_delete.txt`
`touch`	Create an empty file	`touch new_file.txt`
`cat`	Display or concatenate files	`cat file.txt`
`head`	Display the beginning of a file	`head -n 5 file.txt`
`tail`	Display the end of a file	`tail -n 10 file.txt`
`grep`	Search text in files	`grep "pattern" file.txt`
`chmod`	Change file permissions	`chmod 644 file.txt`
`chown`	Change file ownership	`chown user:group file.txt`
`ln`	Create links between files	`ln -s /path/to/file linkname`

Table 1: Important Linux Commands for File and Directory Management[2]

1.2.2 Shell Command

A shell command is a directive or instruction provided by a user to a shell (a command-line interpreter) in an operating system.

1.2.3 Types of Shell Commands

- Internal Commands: These commands are built into the shell itself. They are part of the shell's functionalities and do not exist as separate executable files. Examples include cd, echo, exit, alias, export, etc.
- External Commands: These commands are separate executable files located in directories listed in the system's PATH variable. When a user inputs an external command, the shell searches for the command's executable file in these directories and executes it if found.

1.2.4 Linux File System Hierarchy[2][3]

```
/ (root directory)
├── bin ..... Essential command binaries
├── boot ..... Static files of the boot loader
├── dev ..... Device files
├── etc ..... Host-specific system configuration
│   ├── passwd ..... User information
│   ├── group ..... Group information
│   └── hosts ..... Hosts file
├── home ..... Home directories
│   ├── user1
│   ├── user2
│   └── ...
├── lib ..... Essential shared libraries and kernel modules
├── media ..... Mount points for removable media
├── opt ..... Add-on application software packages
├── proc ..... Kernel and process information pseudo-filesystem
├── sys ..... Kernel and system information
├── tmp ..... Temporary files
├── usr ..... Secondary hierarchy for user data
│   ├── bin ..... Non-essential command binaries
│   ├── include ..... Standard include files
│   ├── lib ..... Libraries for programming
│   └── share ..... Architecture-independent data
└── var ..... Variable data
    ├── log ..... Log files
    └── spool ..... Application spool data
```

1.2.5 Paths

Paths refer to the location or address of a file or directory in the file system.

Types of paths

(i) Absolute Path: An absolute path defines the complete location of a file or directory starting from the root directory (/). It includes the entire directory hierarchy from the root directory to the specific file or directory. For instance, /home/user/documents/file.txt is an absolute

path where the file.txt is located in the 'documents' directory inside the 'user' directory within the 'home' directory, starting from the root (/) directory.

(ii) Relative Path: A relative path defines the location of a file or directory with respect to the current working directory. It doesn't start from the root directory but refers to a location relative to the current directory. For example, if the current directory is /home/user/, a file located in the 'documents' directory can be referenced using a relative path like documents/file.txt.

1.2.6 Linux File Permission Stack

In Linux, file and directory permissions are represented using a permission stack. The format of the permission stack is as follows:

```
- r w x   r - x   r - x
|   | |     | |     | |
|   | |     | |     | +---- Others (permissions for users not covered by owner or group)
|   | |     | +---------- Group (permissions for users in the file's group)
|   | +-------------------- Owner (permissions for the file or directory owner)
|   +---------------------- Type of the file (e.g., - for a regular file, d for a directory)
+------------------------- Special permission bits (e.g., s, t, etc.)
```

Each group of permissions (Owner, Group, Others) consists of three characters representing read (r), write (w), and execute (x) permissions. If a permission is allowed, the respective character is displayed, and if it's denied, a hyphen (-) is shown.

For example:

```
-rw-r--r--  1 user group    24 Jan  7 13:20 myfile.txt
```

In this example: - The first character (-) indicates that it's a regular file. - The next three characters (rw-) represent the owner's permissions (read and write, but not execute). - The following three characters (r–) represent the group's permissions (read-only). - The last three characters (r–) represent permissions for others (read-only).

These permissions can be changed using commands like chmod in Linux to alter the read, write, and execute permissions for the owner, group, and others.

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlve