UNIT- III

Prepared by: Bhavna Barad

❖ Linux Command Introduction

What is Linux?

*Linux is a free and open-source software that operates on its own operating system. The term '**Linux'** stands for **GNU + Linux**. Initially developed by **Linus Torvalds**, it was created alongside the source code of Unix. While Linux is extensively utilized for various purposes, its applications are well-known to many.*

- **Linux commands** are a type of Unix command or **shell** procedure. They are the basic tools used to interact with Linux on an individual level. Linux commands are used to perform a variety of tasks, including displaying information about files and directories.

- Linux operating system is used on servers, desktops, and maybe even your smartphone. It has a lot of command line tools that can be used for virtually everything on the system.

✓ Locating Commands

if you want to locate commands (or programs) on your system, there are several ways to do so. The primary tools used for locating commands or files in the system are *which*, *whereis*, *locate*, and *find*. Here's an overview of these commands and how they work

1. which **Command**

The **`which`** command is used to locate the **executable** of a command. It shows the path of the executable that would be run if the command is typed in the terminal. It only searches directories listed in the **`$PATH`** environment variable.

Syntax:

which <command>

Example:

which ls

Output:

/bin/ls

This shows that the ls command is located in the /bin directory.

---

2. whereis **Command**

The whereis command is more comprehensive than which because it searches for not only the command executable but also related files such as documentation, source code, and man pages. It searches a predefined set of directories, including /bin, /usr/bin, /usr/local/bin, and /usr/share/man.

Syntax:

whereis <command>

Example:

whereis gcc

Output:

gcc: /usr/bin/gcc /usr/share/man/man1/gcc.1.gz

This shows the path to the gcc executable and its manual page.

---

3. *Locate* **Command**

The locate command uses a database to quickly find files on the system. The database is typically updated periodically by the updatedb command (or automatically via a cron job).

locate is very fast because it searches through the pre-built database rather than the actual filesystem.

Syntax:

locate <filename>

Example:

locate vim

Output (sample):

/usr/bin/vim
/usr/share/man/man1/vim.1.gz

In this case, locate found vim in two locations. It's important to note that the locate database may not always be up-to-date, so you might need to run updatedb to update it.

---

4. *find* **Command**

The find command is the most flexible and powerful tool for searching files and directories in a Linux system.

It searches the file system in real-time and can search based on various criteria, such as file name, type, permissions, size, and more.

Syntax:

find <path> -name <pattern>

Example:

find /usr/bin -name vim

Output:

/usr/bin/vim

---

5. *type* **Command**

The type command is used to determine how a command is interpreted by the shell (whether it's a built-in shell command, an alias, or an executable).

This can be useful to understand whether the command is a shell function, an alias, or an actual program.

Syntax:

type <command>

Example:

type ls

Output:

ls is /bin/ls

Summary of Commands

Command	Description	Example Usage
`which`	Locates the **executable** of a command in `$PATH`.	`which ls`
`whereis`	Locates **binary**, **source**, and **manual** files.	`whereis gcc`
`locate`	Locates files using a pre-built database.	`locate vim`
`find`	Searches the file system in real-time with flexible options.	`find /usr/bin -name vim`
`type`	Identifies how a command is interpreted by the shell.	`type ls`

1. Internal Commands (Shell Built-ins)

- Internal commands (also known as built-ins) are commands that are built into the shell itself. They do not require an external program to be executed.

- These commands are part of the shell's internal programming, so when you run them, the shell directly interprets and executes them.

- Internal commands are typically used for controlling the behavior of the shell or for managing shell-related tasks.

Characteristics of Internal Commands:

- They are part of the shell itself (e.g., bash, zsh, sh).
- They execute directly within the shell without launching a separate process.
- They are faster since there is no need to create a new process for execution.
- They typically perform shell management tasks (like managing variables, jobs, or the shell environment).

Examples of Internal Commands:

- *cd* (Change directory)
- *echo* (Print text to the terminal)
- *exit* (Exit the shell)
- *export* (Set environment variables)
- *pwd* (Print working directory)
- *history* (Display the history of commands)
- *set* (Set shell options)
- *unset* (Unset shell variables or functions)
- *type* (Check the type of a command)
- *alias* (Create or display shell aliases)

Example Usage:

```
cd /home/user
echo "Hello, world!"
exit
```

2. External Commands

- External commands are commands that are not built into the shell but instead are separate executable programs stored in files on the filesystem.

- These commands are typically stored in directories listed in the $PATH environment variable.

- When you run an external command, the shell locates the corresponding executable file and runs it as a separate process.

Characteristics of External Commands:

- They are separate executable files located in system directories like /bin, /usr/bin, /sbin, etc.
- They require launching a new process to execute, which makes them slower than internal commands.
- External commands can be written in various programming languages like C, Python, Perl, etc.
- They are used to perform a wide range of tasks that are beyond the shell's internal capabilities, such as file manipulation, system management, network operations, and more.

Examples of External Commands:

- ls (List directory contents)
- cp (Copy files or directories)
- rm (Remove files or directories)
- cat (Concatenate and display files)
- grep (Search text using patterns)
- find (Search for files)
- ping (Send ICMP echo requests to network hosts)
- ps (Display information about running processes)

Example Usage:

```
ls /home/user

cp file1.txt file2.txt

grep "search_term" myfile.txt
```

Key Differences Between Internal and External Commands

Aspect	Internal Commands	External Commands
Location	Part of the shell (built-in)	Stored as executable files on the filesystem
Execution	Executed directly within the shell	Requires launching a new process
Speed	Faster (no need to create a new process)	Slower (new process needs to be created)
Functionality	Primarily for shell control and management	Perform system-level tasks, file operations
Examples	`cd`, `echo`, `exit`, `pwd`, `export`	`ls`, `grep`, `cp`, `find`, `ping`
Usage in Shell	Directly available in any shell session	Must be located in directories listed in `$PATH`

❖ Command Structure

- command structure refers to the syntax and components that make up a command entered into the terminal or shell.

- The structure typically consists of several parts, and each part plays a specific role in how the command is interpreted and executed by the system.

A Linux command typically has the following basic structure:

<command> [options] [arguments]

- ✓ **<command>:** The name of the command or program you want to execute.
- ✓ **[options]:** Optional flags or switches that modify the behavior of the command.
- ✓ **[arguments]:** Optional input, such as files, directories, or other values, that the command operates on.

❖ Linux Basic Commands

1. **mkdir:** Creates a new directory if it does not already exist.

   Syntax**:** mkdir <directory_name>
   Example**:** mkdir my_folder

2. **rmdir:** It is used to delete a directory if it is empty.

   Syntax**:** rmdir <directory_name>
   Example**:** rmdir empty_folder

3. **rm:** Used to remove files or directories.

   Syntax**:** rm <file_name>
   Example**:** rm -r <directory_name>

4. **cp:** This command will copy the files and directories from the source path to the destination path. It can copy a file/directory with the new name to the destination path. It accepts the source file/directory and destination file/directory.

   Syntax**:** cp <source> <destination>

   Example**:** cp -r <source_directory> <destination_directory>

5. **mv:** Used to move the files or directories. This command's working is almost similar to *cp* command but it deletes a copy of the file or directory from the source path.

   Syntax**:** mv <source> <destination>
   Example**:** mv <old_name> <new_name>

6. **ls:** To get the list of all the files or folders.

   Common Options:

   -l: Long listing format (shows detailed info).

   -a: Lists hidden files (files starting with a dot).

   -h: Human-readable file sizes.

   Syntax**:** ls [options] <directory>
   Example**:** ls -l # Lists files with detailed info

7. **cal:** Displays the current month's calendar

   Syntax**:** cal
   Example**:** cal # Displays the current month's calendar

8. **date:** Shows the current date and time.

   Syntax**:** date
   Example**:** date # Displays the current date and time

9. **cat:** It is generally used to concatenate the files. It gives the output on the standard output.

   Syntax**:** cat <file_name>
   Example**:** cat file1.txt # Displays the content of file1.txt

10. **cd:** Used to change the directory.

    Syntax**:** cd <directory_path>
    Example**:** cd /home/user/ # Changes the directory to /home/user
    cd .. # Moves up one directory level

11. **find:** Searches for files and directories in a specified location based on different criteria.

Syntax**:** find <path> <search_criteria>
Example**:** find /home/user/ -name "*.txt" # Finds all .txt files in /home/user/

12. **head:** Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.

Syntax**:** head <file_name>
Example**:** head myfile.txt # Displays the first 10 lines of myfile.txt

13. **tail:** Used to print the last N-1 lines of a file. It accepts N as input and the default value of N is 10.

Syntax**:** trail <file_name>
Example**:** tail myfile.txt
Displays the last 10 lines of myfile.txt

14. **ps:** Lists the currently running processes.

Syntax**:** ps [options]
Example**:** ps aux # Displays all running processes

15. **touch:** Used to create or update a file.

Syntax**:** touch <file_name>
Example**:** touch newfile.txt
Creates an empty file named newfile.txt

16. **sh:** Used to execute shell scripts or start a new shell session.

Syntax**:** sh <script_name>
Example**:** sh myscript.sh
Executes the shell script myscript.sh

17. **who:** Displays a list of users currently logged into the system.

Syntax**:** who
Example**:** who # Lists the users currently logged in

18. **chmod:** Modifies file or directory permissions.

Syntax**:** chmod <permissions> <file_name>

Example**:** chmod +x script.sh # Adds execute permission to script.sh

19. **pwd:** Show the present working directory.

Syntax**:** pwd

20. **echo:** This command helps us move some data, usually text into a file.

Syntax**:** echo <text_or_variable>

Example**:** echo "Hello, world!"
Prints "Hello, world!" to the terminal

echo $HOME
Prints the value of the HOME environment variable

❖ Linux Advanced Commands

1. Finding Patterns in Files (grep, egrep, fgrep, look)

1. **grep:** Searches for a pattern in files and prints lines containing that pattern.

   Syntax**:** grep [options] <pattern> <file_name>

   Example**:** grep "error" logfile.txt

   # Searches for "error" in logfile.txt

   grep -i "warning" logfile.txt

   # Case-insensitive search for "warning"

2. **egrep:** Allows for more complex patterns using extended regular expressions (e.g., +, ?, |).

   Syntax**:** egrep <pattern> <file_name>

   Example**:** egrep "(error|fail)" logfile.txt

   # Searches for "error" or "fail"

3. **fgrep:** Searches for fixed strings (does not support regular expressions)

   Syntax**:** fgrep <string> <file_name>

   Example**:** fgrep "hello" logfile.txt

   # Searches for the exact string "hello"

4. **look:** Finds lines in a sorted file that begin with the specified prefix.

   Syntax**:** look <prefix> <file_name>

   Example**:** look "pre" words.txt # Finds words that start with "pre"

2. Counting Lines, Words and File Size (wc, nl)

1. **wc:** Counts the number of lines, words, and characters in a file.

   Syntax**:** wc [options] <file_name>

   Example**:** wc -l file.txt # Counts the number of lines in file.txt

   wc -w file.txt # Counts the number of words in file.txt

   wc -c file.txt # Counts the number of characters in file.txt

2. **nl:** Adds line numbers to the file's content.

   Syntax**:** nl <file_name>

   Example**:** nl file.txt # Numbers the lines of file.txt

3. Working with Columns and Fields (cut, paste, colrm, join)

1. **cut:** Cut sections from each line of a file.

   Syntax**:** cut [options] <file_name>

   Example**:** cut -d ',' -f 1,3 file.csv

   # Extracts fields 1 and 3 from comma-separated file

2. **paste:** Merge lines of files.

   Syntax**:** paste <file1> <file2>

   Example**:** paste file1.txt file2.txt # Merges lines from file1 and file2

3. **colrm:** Removes columns from the file based on column numbers.

   Syntax**:** colrm <start_column> <end_column>

   Example**:** colrm 5 10 # Removes columns 5 to 10

4. **join:** Join two files based on a common field.

   Syntax**:** join <file1> <file2>

   Example**:** join file1.txt file2.txt # Joins file1 and file2 on their first field

4. Sorting the Contents of Files (sort, uniq)

1. **sort:** Sorts the content of a file in lexicographical or numerical order.

   Options:

   - -n: Sorts numerically.
   - -r: Reverses the order.
   - -u: Removes duplicates.

   Syntax**:** sort [options] <file_name>

   Example**:** sort file.txt # Sorts file.txt in ascending order

   sort -n file.txt # Sorts file.txt numerically

2. **uniq:** Filters out duplicate adjacent lines from a sorted file.

   Syntax**:** uniq [options] <file_name>

   Example**:** uniq file.txt # Removes consecutive duplicate lines

   uniq -c file.txt

   # Displays line counts alongside each unique line

5. Comparing Files (cmp, comm., diff, patch)

1. cmp: Compares two files and reports the first difference.

Syntax**:** cmp <file1> <file2>

Example**:** cmp file1.txt file2.txt

Compares file1.txt and file2.txt byte by byte

2. comm.: Displays lines that are common and different between two sorted files.

Syntax**:** comm <file1> <file2>

Example**:** comm file1.txt file2.txt

Shows common and unique lines between file1 and file2

3. diff: Shows differences between two files.

Syntax**:** diff <file1> <file2>

Example**:** diff file1.txt file2.txt

Displays line-by-line differences between file1 and file2

4. patch: Applies a patch (set of differences) to a file.

Syntax**:** patch **<** <patch_file>

Example**:** patch < changes.diff

Applies the patch file changes.diff to the target file

6. Changing Information in files (tr)

1. **tr:** Translate or delete characters

   Syntax**:** tr [options] <set1> <set2>

   Example**:** tr 'a-z' 'A-Z' < file.txt

   # Converts lowercase to uppercase in file.txt

7. Performing Mathematical Calculations (bc, dc)

1. **bc:** An interactive calculator that supports precision arithmetic.

   Syntax**:** bc

   Example**:** echo "3 + 5" | bc # Outputs 8

2. **dc:** A command-line calculator that uses reverse Polish notation.

   Syntax**:** dc

   Example**:** echo "3 5 + p" | dc # Outputs 8 (3 + 5)

Summary Table

Command	Purpose	Example
`grep`	Search for patterns in files using regex	`grep "error" file.txt`
`egrep`	Extended grep for advanced regex	`egrep "(error`
`fgrep`	Fixed-string grep, no regex	`fgrep "hello" file.txt`
`look`	Find lines starting with a string	`look "pre" words.txt`
`wc`	Count lines, words, or characters in a file	`wc -l file.txt`
`nl`	Number the lines in a file	`nl file.txt`
`cut`	Cut sections of lines by delimiter or position	`cut -d ',' -f 1,3 file.csv`
`paste`	Merge lines of files side by side	`paste file1.txt file2.txt`
`colrm`	Remove columns from a file	`colrm 5 10`
`join`	Join files based on a common field	`join file1.txt file2.txt`
`sort`	Sort lines of a file	`sort file.txt`
`uniq`	Remove duplicate adjacent lines from a file	`uniq file.txt`
`cmp`	Compare files byte by byte	`cmp file1.txt file2.txt`
`comm`	Compare two sorted files line by line	`comm file1.txt file2.txt`
`diff`	Compare files line by line	`diff file1.txt file2.txt`
`patch`	Apply a patch (differences) to a file	`patch < changes.diff`
`tr`	Translate or delete characters	`tr 'a-z' 'A-Z' < file.txt`
`bc`	Command-line calculator	`echo "3 + 5"`
`dc`	Reverse Polish Notation calculator	`echo "3 5 + p"`

Thank You