OPERATING SYSTEM

UNIT-4

Introduction to Linux Operating System:

Definition: Linux is a free and open-source operating system kernel initially developed by Linus Torvalds in 1991. It is based on the Unix operating system and is widely used in servers, desktops, embedded systems, and supercomputers.

Key Features of Linux:

1. Open Source: Linux is distributed under open-source licenses, allowing users to modify and distribute the source code freely.
2. Multiuser: Supports multiple users accessing the system simultaneously with different privileges and permissions.
3. Multitasking: Capable of running multiple processes concurrently, allowing efficient utilization of system resources.
4. Multithreading: Supports multithreading, enabling multiple threads of execution within a single process.
5. Security: Provides robust security features, including user authentication, file permissions, and encryption.
6. Stability: Known for its stability and reliability, with long uptimes and minimal system crashes.
7. Flexibility: Offers a wide range of software packages and customization options to suit various user needs.
8. Portability: Runs on a variety of hardware architectures, making it suitable for a diverse range of devices.

Basic Utilities in Linux:

1. Shell: The command-line interface used to interact with the Linux system. Common shells include Bash, Zsh, and Fish.
2. File Management: Utilities for creating, copying, moving, and deleting files and directories, such as `mkdir`, `cp`, `mv`, and `rm`.
3. Text Processing: Tools for working with text files, including `cat`, `grep`, `sed`, and `awk`, for viewing, searching, and manipulating text.

4. Process Management: Utilities for managing processes, such as `ps`, `top`, `kill`, and `nohup`, for listing running processes, terminating processes, and running processes in the background.
5. Package Management: Tools for installing, updating, and removing software packages, such as `apt` (Advanced Package Tool) on Debian-based systems and `yum` on Red Hat-based systems.
6. Networking: Utilities for networking tasks, including `ping`, `ifconfig`, `netstat`, and `ssh`, for network troubleshooting, configuration, and remote access.
7. System Administration: Tools for system administration tasks, such as `sudo`, `su`, `useradd`, and `passwd`, for managing users, permissions, and system configuration.

Working with Files in Linux:

1. File Permissions: Each file in Linux has permissions that control who can read, write, and execute the file. Permissions are represented by three sets of characters: owner, group, and others.
2. File Ownership: Each file is associated with an owner and a group. Owners have full control over their files, while group members have permissions specified by the group.
3. File Operations:
    - Creating Files and Directories: Use the `touch` command to create empty files and `mkdir` to create directories.
    - Viewing Files: Use commands like `cat`, `less`, and `tail` to view the contents of files.
    - Editing Files: Use text editors like `vi`, `nano`, or `gedit` to edit files.
    - Copying and Moving Files: Use the `cp` command to copy files and `mv` to move or rename files.
    - Deleting Files: Use the `rm` command to delete files. Be cautious as this action is irreversible.
4. File Searching: Utilize commands like `find` and `locate` to search for files based on their names, types, or content.
5. File Compression: Use utilities like `gzip`, `bzip2`, and `tar` to compress and decompress files and directories.

Example:

Suppose you're a student working on a Linux-based research project. You use the command line extensively to perform various tasks:

1. Creating Files and Directories: You create a new directory for your project using `mkdir project`, and within it, you create files for your code using `touch main.c` and `touch functions.h`.
2. Editing Files: You use the `vi` text editor to write and edit your code files. For instance, you open `main.c` using `vi main.c` and make changes to the code.
3. Compiling Code: After writing your code, you compile it using the `gcc` compiler, such as `gcc main.c -o program`.
4. Running Programs: You execute your compiled program using `./program`, observing the output for debugging or validation purposes.
5. Managing Files: Throughout your project, you copy files between directories, move files to organize them, and delete unnecessary files using commands like `cp`, `mv`, and `rm`.
6. Networking: You use SSH (`ssh`) to access remote servers for collaboration or to transfer files securely.
7. Process Management: You monitor the system's resource usage and manage running processes using commands like `ps`, `top`, and `kill` to ensure optimal performance.

Shells in Linux:

Definition: A shell is a command-line interpreter that provides a user interface for access to an operating system's services. It accepts commands from the user and executes them by interacting with the operating system kernel.

Key Functions of Shells:

1. Command Interpretation: Interprets and executes user commands entered via the command line.
2. Environment Management: Manages environment variables, which are dynamic-named values that affect the behavior of running processes.
3. Script Execution: Executes shell scripts, which are sequences of shell commands stored in a file.
4. Input/Output Redirection: Redirects input and output streams to and from files or other processes.
5. Job Control: Manages running processes, allowing users to start, stop, suspend, and resume processes.

Types of Shells in Linux:

1. Bourne Shell (sh): The original Unix shell, developed by Stephen Bourne. Simple and efficient, but lacks some advanced features found in modern shells.

2. Bourne-Again Shell (bash): The default shell on most Linux distributions. An enhanced version of the Bourne shell with additional features like command-line editing, history, and job control.
3. C Shell (csh): Designed to resemble the C programming language syntax. Features interactive command-line editing and history similar to bash.
4. Korn Shell (ksh): Developed by David Korn as a superset of the Bourne shell. Offers advanced scripting capabilities and features found in both the Bourne and C shells.
5. Z Shell (zsh): A powerful and feature-rich shell with advanced command-line editing, spell-checking, and plugin support. Known for its extensibility and customization options.
6. Fish Shell (fish): Known for its user-friendly syntax highlighting, autosuggestions, and web-based configuration interface. Designed to be intuitive and easy to use for both beginners and experienced users.

Introduction to Shell Programming:

Definition: Shell programming involves writing scripts to automate tasks or perform system administration tasks using shell commands. Shell scripts are text files containing sequences of shell commands that are executed by the shell interpreter.

Key Concepts in Shell Programming:

1. Variables: Used to store values that can be manipulated or referenced within a script.
2. Control Structures: Constructs like loops and conditional statements for controlling the flow of execution.
3. Functions: Blocks of reusable code that perform specific tasks.
4. Input/Output Redirection: Redirecting input and output streams to and from files or other processes.
5. Command Substitution: Embedding the output of a command into another command or expression.

Editors in Linux:

1. vi/vim:
    - Vi is a classic text editor available on most Unix-like systems.
    - Vim (Vi Improved) is an enhanced version of Vi with additional features and improvements.
    - Known for its modal editing capabilities, allowing users to switch between different modes for inserting text, navigating, and editing.
2. Emacs:
    - A highly extensible text editor known for its powerful editing features and support for various programming languages.

    - Provides a wide range of functionalities, including syntax highlighting, auto-completion, and built-in documentation.
3. nano:
    - A simple and user-friendly text editor suitable for beginners.
    - Provides basic editing functionalities similar to traditional text editors like Notepad.
4. gedit:
    - The default text editor for the GNOME desktop environment on Linux.
    - Features a clean and intuitive interface with support for syntax highlighting and plugins.
5. Sublime Text:
    - A popular cross-platform text editor known for its speed, flexibility, and extensive plugin ecosystem.
    - Offers advanced editing features like multiple selections, split editing, and command palette.
6. Atom:
    - A modern and customizable text editor developed by GitHub.
    - Built using web technologies, allowing for easy customization and extension with plugins.

Example:

Suppose you need to write a shell script to automate the backup of important files on your Linux system. You decide to use bash as your shell and vim as your text editor:

1. Creating a Shell Script: You use vim to create a new file named `backup.sh` and start writing your shell script.
2. Writing the Script: You define variables to specify the source directory containing files to be backed up and the destination directory for storing backups. You then use shell commands like `cp` to copy files from the source directory to the destination directory.
3. Executing the Script: After saving the script, you make it executable using the `chmod` command (`chmod +x backup.sh`). You can then run the script using `./backup.sh` to initiate the backup process.

Vim is a powerful and versatile text editor that is widely used in the Unix and Linux community. It stands for "Vi IMproved" and is an enhanced version of the classic Vi text editor. Vim is known for its efficient editing capabilities, extensive feature set, and high degree of customizability. Let's delve into an introduction to Vim:

Key Features of Vim:

1. Modal Editing:
    - Vim operates in different modes: Normal mode, Insert mode, Visual mode, and Command-line mode.
    - Normal mode is used for navigation and executing commands.
    - Insert mode is for inserting text.
    - Visual mode is for selecting text.
    - Command-line mode is for entering commands.
2. Efficient Navigation:
    - Vim provides various keyboard shortcuts for efficient navigation, such as h, j, k, l for moving left, down, up, and right respectively.
    - Advanced navigation commands include word-wise movement, line-wise movement, and jumping to specific lines or characters.
3. Extensive Editing Commands:
    - Vim offers a wide range of editing commands for manipulating text, including deleting, copying, pasting, replacing, and joining lines.
    - Commands can be combined with movement commands to operate on specific ranges of text.
4. Customization and Plugins:
    - Vim is highly customizable, allowing users to configure various settings, key mappings, and color schemes to suit their preferences.
    - It supports a rich ecosystem of plugins that extend its functionality for tasks such as code editing, version control, and file management.
5. Syntax Highlighting and Auto-Completion:
    - Vim provides syntax highlighting for various programming languages and file types, making code easier to read and understand.
    - It offers auto-completion features to assist users in typing commands and programming constructs.
6. Built-in Help System:
    - Vim includes a comprehensive built-in help system accessible through the :help command, providing documentation and explanations for Vim commands and features.

Getting Started with Vim:

1. Opening Vim:
    - To open Vim, simply type `vim` followed by the name of the file you want to edit (`vim filename`). If the file does not exist, Vim will create a new file with that name.
2. Modes in Vim:
    - Vim starts in Normal mode by default. Press `i` to enter Insert mode and start typing text. Press `Esc` to return to Normal mode.

- Use `:` to enter Command-line mode for executing commands or saving/quitting files.

3. Basic Navigation:
    - Use arrow keys or `h`, `j`, `k`, `l` keys for navigation in Normal mode.
    - Other navigation commands include `w` (move forward one word), `b` (move backward one word), `0` (move to the beginning of the line), and `$` (move to the end of the line).
4. Basic Editing:
    - In Normal mode, use `x` to delete characters, `dd` to delete lines, `yy` to copy lines, and `p` to paste copied or deleted text.
5. Saving and Quitting:
    - To save changes and quit Vim, type `:wq` in Command-line mode and press `Enter`.
    - To quit without saving changes, type `:q!` and press `Enter`.

Example:

Let's say you want to edit a text file named `example.txt` using Vim:

1. Opening the File:
    - Open a terminal window and navigate to the directory containing `example.txt`.
    - Type `vim example.txt` and press `Enter` to open the file in Vim.
2. Editing the File:
    - Use navigation keys to move around the file and `i` to enter Insert mode.
    - Make changes to the text as needed.
3. Saving and Quitting:
    - Press `Esc` to return to Normal mode, then type `:wq` and press `Enter` to save changes and quit Vim.
    -