Navigating and manipulating the Linux Filesystem

Filesystem

The filesystem is the backbone of any operating system, and without it, data cannot be organized or accessed systematically. It continually evolves to keep up with increasing storage needs (managing how data is stored and retrieved) and to ensure data security, making it an essential part of the digital infrastructure in every computer.

Filesystem Function

- **Storage**: Defines how data is stored and organized into files and folders.
- **Retrieval**: Determines how to access the stored files and locate them on the disk.
- **Naming**: Specifies how files and folders are named, including whether they contain extensions or other attributes.
- **Structure**: Creates a tree or hierarchical structure for files and folders to organize data in a way that is easily accessible.

Key Components of a Filesystem

- **Blocks**: Data storage units on disks. The storage space is divided into multiple small-sized blocks, each containing a portion of the data.
- **Inodes**: Parts of the filesystem that contain information about files, such as the file name, size, timestamps (e.g., creation date), and the actual location of the data stored in blocks.
- **Path**: The address or method by which a specific file within the file structure can be accessed. The path can be either absolute or relative.

- **Files and Folders**: Files are units of data, while folders contain a collection of files or subfolders to organize the files.

Types of Filesystems

- **FAT32**: The File Allocation Table (FAT) filesystem is common in Windows systems. It is simple but faces limitations such as the maximum file size (4GB).
- **NTFS**: The New Technology File System used in modern Windows systems. It offers features like security and permissions management, file compression, and support for large files.
- **EXT4**: The Fourth Extended Filesystem, most common in Linux systems, provides high reliability and performance in data management.
- **APFS**: The Apple File System used in macOS, offering faster and more efficient file handling, with features such as snapshots and encryption.

Filesystem Hierarchy

Files in the operating system are organized in a tree structure that starts with a root directory ("/"). From the root, folders and files branch out. For example, in Unix and Linux systems, the file structure is arranged in a hierarchy that starts from "/" as the root of all files and folders.

Filesystem Operations

- **Creating Files**: A name and storage space on the disk are allocated, and the inodes are updated.
- **Reading Files**: The OS parses the file path, accesses the appropriate inode, and then loads the blocks containing the data.

- **Modifying Files**: When a file is modified, the system writes the new data into the blocks and updates the inodes.
- **Deleting Files**: The inodes associated with the file are removed, and the blocks allocated to it are freed.

Filesystem Challenges

- **Fragmentation**: Over time, files are scattered across multiple, non-contiguous blocks, affecting performance and slowing down file reads.
- **Cross-System Compatibility**: Different filesystems are not always compatible with all operating systems, such as NTFS and EXT4, which require additional tools to access data across systems.

Permissions and Security

The filesystem is also responsible for managing file and folder permissions, specifying who can read, modify, or execute files. This is done by assigning permissions to users and groups, as in the Linux permission system (**rwx - Read, Write, Execute**).

Recent Filesystem Enhancements

- **Encryption**: Maintaining data confidentiality through encryption techniques at the filesystem level.
- **Snapshots**: Creating recoverable copies of files at a specific point in time.
- **Failure Recovery**: Modern systems like ZFS offer automatic error correction and data recovery features.

Filesystem Components in Linux

- **/** (root): The highest level in the filesystem, containing all folders and files in the system.
- **/home**: Contains user directories. Each user has their own directory here.
- **/etc**: Contains system configuration files.
- **/var**: Stores logs and other variable files.
- **/tmp**: Contains temporary files created by the system and programs.
- **/bin**: Contains essential commands required by the system, such as ls, cp, mv, rm.
- **/usr**: Contains programs and libraries for users.

Commands for Navigating the Filesystem

- **pwd (Print Working Directory)**: Displays the full path of the current directory you are working in.
- **ls (List)**: Displays a list of files and folders in the current directory.
    - **ls -l**: Displays files with details such as permissions, size, and date (l stands for long).
    - **ls -a**: Displays hidden files (those starting with a dot).
- **cd (Change Directory)**: Used to navigate between directories.
    - **cd /home/kali** : Move to the user's documents folder.
    - **cd ..** : Go back to the previous folder.
    - **cd /**: Navigate to the root of the filesystem.

- **cd ~**: Move to the current user's home directory (~ called Tilde sign).

- **tree**: Displays the directory and file structure in a tree format.

Creating Directories and Files

- **mkdir (Make Directory)**: Creates directories.
  - **mkdir directory_name**: Creates a single directory.
  - **mkdir -p Desktop/A1/A2**: Creates multiple nested directories (-p stands for parent).
  - **mkdir -m: The -m option specifies the permissions for the new directory. Permissions can be set using numeric (octal) or symbolic format.**
- **touch**: Creates files.
  - **touch filename.txt**: Creates a new, empty file.
  - **touch file1.txt file2.txt file3.txt**: Creates multiple files at once.
- **echo**: Creates a file and adds content to it.
  - **echo "Text to add" > filename.txt**: Creates a file and adds text to it.
  - **echo "Text to add" >> filename.txt**: Adds text to an existing file.

Viewing and Editing Files

- **cat (Concatenate)**: Displays the contents of a file or combines multiple files into one.

- **cat filename.txt**: Displays the content of a file.
- **cat file1.txt file2.txt > file3.txt**: Combines multiple files into one.
- **cat > newfile.txt**: Creates a new multi-line file. You can add text and press Ctrl+D to save.
- **cat file2.txt >> existingfile.txt**: Appends the content of one file to another.

- **more**: Displays file contents page by page. You can navigate using space or arrow keys.
  - **more filename.txt**: Displays text starting from the beginning.
    - Pressing Space shows the next page.
    - Pressing Enter moves one line at a time.
    - Press q to exit.
- **less**: Similar to more, but with more advanced navigation, allowing scrolling both forward and backward.
  - **less filename.txt**: Displays file content with advanced navigation.
    - You can move up and down using the arrow keys.
    - Press Space to move to the next page, b to go back, and q to quit.

- **nano**: Opens or creates a text file using the Nano text editor.
  - **nano filename.txt**: Opens a file for editing.

Searching and Deleting Files

- **grep (Global Regular Expression Print)**: A powerful tool used to search for a specific pattern in files or text.
    - **grep "pattern" file.txt**: Searches for a string in the file.
    - **grep -i "hello" file.txt**: Case-insensitive search.
    - **grep "error" *.txt**: Searches in multiple files.
    - **grep -c "error" log.txt**: Displays the count of occurrences of the pattern.
- **rm**: Deletes files or directories.
    - **rm filename.txt**: Deletes a file.
    - **rmdir foldername**: Deletes an empty directory.
    - **rm -r foldername**: Deletes a non-empty directory and its contents.

Copying and Moving Files

- **cp**: Copies files and directories.
    - **cp source_file destination_directory**: Copies a file.
    - **cp -r source_directory destination_directory**: Recursively copies a directory and its contents.
- **mv**: Moves files and directories.
    - **mv oldname.txt newname.txt**: Renames a file.
    - **mv myfolder /home/user/documents/**: Moves a directory.

Piping in Linux

Piping is a technique that allows connecting the output of one command as the input to another command. This is a powerful way to chain multiple commands together to execute more complex tasks using small, simple tools.

- The pipe symbol **|** is used to connect commands.
- When using pipes, the output of the first command is directly passed as input to the second command.

Example:

- **cat file.txt | grep "pattern"**: Displays the contents of a file and pipes it to grep to search for a pattern.
  - cat file.txt: Displays the content of the file.
  - grep "pattern": Searches for the desired pattern in the output.
- Filtering large outputs with ls and more:
  - **ls -l | more**: Displays the output of ls -l (which shows a detailed list of files and directories), then uses more to break the output into pages for easier reading.

Basic Commands for System and Information Management

- **uname**: Displays basic system information.
  - **uname -a**: Shows all basic system information.
  - **uname -r**: Shows the kernel version.
- **df (Disk Free)**: Displays information about available and used disk space.

- **df -h**: Shows the disk space in human-readable format (MB, GB).

- **du (Disk Usage)**: Displays disk usage by files and directories.
  - **du -h** /path/to/directory: Displays the disk usage of a directory in human-readable format.
- **lsblk (List Block Devices)**: Lists the structure of disks connected to your system, including partitions and space.
  - **lsblk**: Displays block devices.
- **fdisk**: Displays partition information on the hard disk.
  - **fdisk -l**: Lists partition tables.

Managing Processes and System Information

- **ps (Process Status)**: Displays a list of running processes.
  - **ps**: Shows running processes.
  - **ps** aux: Shows more detailed process information.
    - a: Shows processes of all users.
    - u: Shows processes in detailed format (including the user running the process).
    - x: Shows processes not attached to a terminal.
- **top**: Displays real-time information about running processes, including CPU and memory usage.
  - **top**: Shows running processes in real-time. Press q to exit.

- **kill**: Terminates processes using the process ID (PID).
    - **kill [PID]:** Terminates a specific process.
    - **killall [process_name]:** Terminates all processes with the same name.
        - killall firefox: Closes all Firefox windows.

Home Work:

- Explain the role of inodes in the Linux filesystem. What information do they store, and how are they crucial for file management?
- Describe the differences between the FAT32, NTFS, EXT4, and APFS filesystems. Include a discussion of their primary use cases and limitations.
- Using the Linux command line, demonstrate how to create a new directory structure with multiple nested folders and set specific permissions for the new directories.
- What is the purpose of the grep command in Linux? Provide an example of how you would use it to search for a pattern within multiple text files.
- Explain how piping works in Linux. Give an example where you combine the output of the cat command with grep to filter file content.