Linux

Working with Files

Introduction

Linux is a powerful, open-source operating system that is widely used in servers, embedded systems, and desktops. Understanding how to work with files in Linux is a fundamental skill for students, developers, and system administrators.

The Linux Filesystem Structure

1. Root Directory (/)

- The root directory is the starting point of the Linux filesystem hierarchy.
- All other files and directories stem from the root, forming a tree-like structure.
- It is denoted by a single forward slash (/).

- Only the root user (system administrator) has full access to all contents under /.

2. /home

- This directory stores personal directories for each user on the system.
- For example, user john will have /home/john containing personal files, desktop settings, downloads, etc.
- Users have read/write access only to their own home directories.
- Helps in organizing user data separately and securely.

3. /etc

- Contains system-wide configuration files.
- Files like /etc/passwd, /etc/fstab, and /etc/hosts live here.

- Configuration for services such as networking, firewall, and user management is stored here.
- It is readable by all users, but only writable by the root user.

4. /var

- Stands for "variable", used for files that change frequently.
- Includes:
  - /var/log/: system and application log files
  - /var/mail/: user mailbox files
  - /var/spool/: spool directories for tasks like print queues
  - /var/tmp/: temporary files that are preserved between system reboots.
- Essential for monitoring and maintaining system health and logs.

5. /bin and /sbin

- Both contain essential executable binaries:

  /bin:

  - Holds basic command binaries needed by all users and during boot (e.g., ls, cp, mv, bash).
  - These tools are available in single-user mode.

  /sbin:

  - Contains system binaries used primarily by the system administrator.
  - Examples include ifconfig, reboot, fsck, etc.
  - These are critical for system booting, maintenance, and repair.

Basic File Operations

1. Creating Files

1.1 touch filename

- **Purpose:**

  The touch command is used to create an empty file if it doesn't already exist, or to update the access and modification timestamps of an existing file without changing its contents.

- **Use case:**
  - Quickly creating placeholder files.
  - Initializing log files before use.
  - Updating timestamps to indicate recent access or modification, which can be helpful in scripting or automation.

Example:

```
touch notes.txt
```

- If notes.txt does not exist, this will create an empty file named notes.txt.

- If notes.txt already exists, it will simply update its last modified time to the current time.

1.2 echo "text" > filename

- **Purpose:**

  The echo command outputs the specified text to the terminal by default, but when combined with the > operator, it creates a new file and writes the given text into it.

  If the file already exists, its content will be overwritten with the new text.

- **Use case:**
  - Create a file with some initial content.

    - Overwrite the contents of an existing file with new data.
    - Use in scripts to write logs or output to a file automatically.

Example:

```
echo "Hello World" > hello.txt
```

- Creates a file named hello.txt and writes the text Hello World into it.
- If hello.txt already exists, its contents will be replaced with Hello World.

2. Viewing Files

2.1 cat filename

- **Purpose:**

  The cat (short for concatenate) command is used to display the entire content of a file directly in the terminal window. It can also be used to combine and output the contents of multiple files.

- **Limitations:**

  While cat is fast and convenient, it is not ideal for viewing large files, as the content will scroll past quickly without giving you a chance to read everything. It's best suited for short files or when you're piping content into other commands.

- **Use case:**
  - Quickly checking the contents of small files.
  - Concatenating multiple files together.
  - Using in shell scripts for file output display.

Example:

```
cat hello.txt
```

- This command displays the full content of the file hello.txt in the terminal window.

2.2 less filename

- Purpose:

  The less command is used to view the content of large files one screen (or page) at a time. It's more powerful and user-friendly than cat when dealing with lengthy files, as it allows scrolling and searching.

- Features:
  - Arrow keys or Page Up/Page Down: Scroll through the file.
  - /text: Search forward for the word "text".
  - n: Repeat the last search.
  - q: Quit and return to the terminal.

    - Does not load the whole file into memory, making it ideal for very large files.

- Use Case:
    - Reading large configuration or log files.
    - Navigating through documentation or code files.
    - You need to search for specific text within the file.

Example:

```
less notes.txt
```

- Opens the notes.txt file in a scrollable viewer.
- You can navigate through the content, search with /, and press q to exit.

2.3 more filename

- **Purpose:**

  The more command is used to view the contents of a file one screen at a time, similar to less. However, it is more limited in functionality, as it allows only forward navigation—you cannot scroll back up.

- **Limitations:**
  - You cannot scroll backward in the file.
  - Fewer navigation and search features compared to less.

- **Use Case:**
  - Quickly reading through medium-sized files.
  - Environments where less may not be available.
  - Viewing file output in scripts or on systems with minimal tools.

- **Navigation:**
    - Press Enter to scroll line by line.
    - Press Spacebar to scroll page by page.
    - Press q to quit.

Example:

```
more notes.txt
```

- Displays the notes.txt file one screen at a time, starting from the top.

3. Editing Files

3.1 nano filename

- **Purpose:**

    The nano command opens the Nano text editor, a simple and beginner-friendly terminal-based text editor. It is widely used because it's easy to use and does not require advanced knowledge of commands.

- **Navigation:**
  - Arrow keys: Move the cursor around the text.
  - Arrow keys: Move the cursor around the text.
  - Ctrl + X: Exit the editor. It will ask to save if you have unsaved changes.
  - Ctrl + K: Cut a line.
  - Ctrl + U: Paste a line.
  - Ctrl + W: Search within the file.

- **Use Case:**
  - Quickly editing configuration files.
  - Writing or updating short scripts.
  - Beginners learning to work in the terminal.

Example:

```
nano notes.txt
```

- Opens notes.txt in the Nano editor.
- If the file doesn't exist, Nano will create it.

3.2 vim filename

- **Purpose:**

  The vim command opens the Vim text editor, a powerful and highly configurable terminal-based editor preferred by many advanced users for programming and system administration tasks.

- **Modes in Vim:**

  Vim operates in multiple modes. Understanding these is key to using Vim effectively:

    - Insert Mode (i): Press i to start typing or inserting text.
    - Command Mode (Esc): Press Esc to enter command mode, where you can run commands like save or quit.

- **Navigation:**
    - :w : Save the file.
    - :q: Quit Vim.
    - :wq : Save and quit.
    - :q! : Quit without saving changes.
    - u : Undo last change.

- **Use Case:**
    - **Advanced text editing** and programming.
    - Editing multiple files with tabs or splits.
    - When you want speed, power, and customization.

Example:

```
vim notes.txt
```

- Opens the notes.txt file in Vim.
- If the file doesn't exist, Vim will create it.
- Press i to begin editing, Esc to stop, and :wq to save and exit.

3.3 gedit filename

- Purpose:

  The gedit command opens Gedit, a graphical text editor designed for the GNOME desktop environment. It provides a clean, user-friendly interface, making it suitable for users who prefer a point-and-click editing experience.

- Note:
    - gedit only works on systems that have a graphical user interface (GUI) installed.
    - It will not work on headless servers (i.e., systems without a GUI).

- **Features:**
    - Syntax highlighting for various programming languages.
    - Tabbed interface to open and edit multiple files.
    - Supports copy-paste, undo-redo, and find-replace.
    - Ideal for editing scripts, notes, and configuration files in a desktop setup.

- **Use Case:**
    - New users who prefer a graphical over a terminal-based editor.
    - Simple and clean editing of code, scripts, or documents in GUI environments.

Example:

```
gedit notes.txt
```

- Opens the file notes.txt in the Gedit graphical text editor.
- If the file doesn't exist, it will be created.

4. Copying Files

4.1 cp source destination

- **Purpose:**

  The cp command is used to copy files and directories from one location to another. It can handle single files, multiple files, or entire directory structures depending on the options used.

- **Basic Syntax**:

```
cp [options] source destination
```

- **Common Options:**
    - `-r` or `--recursive`: Used to copy directories and their contents recursively.
    - `-v` or `--verbose`: Displays detailed output of what is being copied, useful for tracking multiple file operations.
    - `-i`: Prompts for confirmation before overwriting existing files.
    - `-u`: Copies only when the source file is newer than the destination or if the destination file is missing.

- **Use Case:**
    - Backing up files or directories.
    - Duplicating configuration files before editing.
    - Moving content from one folder to another without affecting the original.

Examples:

cp file1.txt file2.txt	Copy file1.txt to file2.txt
cp -r folder1/ backup_folder/	Recursively copy folder1 into backup_folder
cp -v report.txt /home/user/	Copy with progress display

5. Moving or Renaming Files

5.1 mv source destination

- **Purpose:**

  The `mv` command is used to move or rename files and directories in Linux. It changes the location or name of a file without creating a copy.

- **Behavior:**
  - If the destination is a filename, `mv` renames the file.
  - If the destination is a directory, `mv` moves the file into that directory.

  - Existing files at the destination may be overwritten without warning (use `-i` for prompt).

- **Common Options:**
  - `-i`: Prompts before overwriting an existing file.
  - `-v`: Shows what's being moved or renamed (verbose mode).
  - `-u`: Moves only if the source is newer or the destination doesn't exist.

- **Use Case:**
  - Organizing files into folders.
  - Renaming files or directories.
  - Replacing outdated files with new ones.

Examples:

mv oldname.txt newname.txt	Renames the file
mv notes.txt /home/user/documents/	Move file to another directory
mv -iv draft.txt final.txt	Rename with confirmation and verbose output

6. Deleting Files and Directories

6.1 rm filename

- **Purpose:**

  The `rm` command is used to **remove (delete) files and directories** permanently from the filesystem.

- **Important Note:**

  Deleted files **cannot be recovered** easily, so use this command with caution.

- **Common Options:**
  - `-r`: Recursively delete a directory and all its contents (subdirectories and files).
  - `-f`: Force deletion without prompting for confirmation, even if files are write-protected.
  - `-i`: Prompt for confirmation before deleting each file.

- **Use Case:**
    - Remove unwanted files or directories to free up space.
    - Clean up temporary files or logs.
    - Permanently delete sensitive data (with care).

Examples:

rm file.txt	Delete a single file
rm -r folder/	Delete a folder and its contents
rm -rf folder/	Forcefully delete a folder and its contents
rm -i important.txt	Prompt before deleting the file