Linux Files and Directories
1. Basics of Files and Linux File System

Introduction to Files:

A file is a collection of data or information stored on a computer. It can contain various types of data, such as text, images, videos, or program instructions. Files are essential for storing and organizing data on a computer system, enabling data persistence and easy access.

Linux File System:

The Linux File System is a hierarchical structure that organizes and manages files and directories (folders) in Linux-based operating systems. The file system provides a unified way of accessing and managing data stored on different storage devices, such as hard drives, SSDs, and external drives.

File Paths:

In Linux, a file path is a textual representation of a file's location within the file system hierarchy. It starts from the root directory ("/") and navigates through directories until reaching the desired file. Paths can be absolute (starting from the root) or relative (starting from the current working directory).

Directories (Folders):

Directories, also known as folders, are special types of files that can contain other files and directories. They are used to organize files hierarchically, making it easier to manage and locate data.

Working Directory:

The working directory is the directory in which the user is currently operating or executing commands. When you open a terminal, you are typically in your home directory, and you can navigate to different directories using commands like cd (change directory).

File Permissions:

Linux uses a permission system to regulate access to files and directories. There are three types of permissions: read, write, and execute. Permissions can be set for three groups: owner, group, and others. The **chmod** command is used to change file permissions.

Basic File Operations:

Common file operations in Linux include:

Creating a new file: **touch** filename

Example:

touch example.txt

Copying files: **cp** sourcefile destination

Example:

cp source.txt destination.txt

Moving files (also used for renaming):

Moving files: **mv** sourcefile destination (also used for renaming)

Example:

mv oldname.txt newname.txt

mv file.txt /path/to/new/location/

Removing files: **rm** filename

Example:

rm unwanted_file.txt

Listing files: **ls**

Example:

ls

Listing files with detailed information, including permissions and ownership:

ls -l

Listing all files, including hidden files:

ls -a

File Ownership:

Every file in Linux is associated with an owner and a group. The chown command is used to change the file's owner, while the chgrp command is used to change its group ownership.

Example:

chmod 644 example.txt # Sets read and write permissions for the owner and read-only for the group and others

Changing file ownership:

Example:

chown user:group example.txt

Changing group ownership:

Example:

chgrp newgroup example.txt

File System Navigation:

To navigate the Linux file system, you can use commands like:

cd: Change directory

Example:

cd /path/to/directory/

pwd: Print working directory

Example:

pwd

cd ..: Move up one level in the directory hierarchy

File Extensions:

Unlike some other operating systems, ***Linux does not rely heavily on file extensions to determine file types***. Instead, it uses file permissions and file content to identify the file type. However, file extensions are still commonly used to give users a hint about the file's content or intended use.

Understanding the basics of files and the Linux file system is crucial for effectively managing data, performing operations, and maintaining the security and integrity of your system. Learning more advanced file system concepts can greatly enhance your proficiency in Linux administration and usage.

2. File Permissions

- File permissions in Linux determine who can perform various operations (read, write, execute) on a file.
- Each file has three sets of permissions: one for the owner, one for the group, and one for others (everyone else).
- These permissions can be represented using a combination of letters and numbers.

File Permission Representation:

Read: r (denoted by the number 4)

Write: w (denoted by the number 2)

Execute: x (denoted by the number 1)

Numerical Representation:

Each set of permissions (owner, group, others) can be represented by a three-digit number, where each digit represents the sum of the corresponding permissions. For example:

0: No permissions (---)

1: Execute-only (--x)

2: Write-only (-w-)

3: Write and execute (-wx)

4: Read-only (r--)

5: Read and execute (r-x)

6: Read and write (rw-)

7: Read, write, and execute (rwx)

Viewing File Permissions:

To view the permissions of files in the current directory, you can use the ls -l command. It will display a list of files along with their permissions, ownership, and other details. For example:

ls -l

-rw-r--r-- 1 user group 1024 Aug 3 10:00 example.txt

Changing File Permissions:

The chmod command is used to change file permissions. There are two ways to use chmod:

Symbolic notation: This method uses symbols (+, −, =) to add, remove, or set permissions. For example:

chmod +x script.sh # Adds execute permission for all (owner, group, others)

chmod u+w file.txt # Adds write permission for the owner

chmod go-r file.txt # Removes read permission for the group and others

chmod o=rw file.txt # Sets read and write permissions for others

Octal notation: This method uses the numerical representation (0–7) to specify permissions directly. For example:

chmod 644 example.txt # Sets read and write permissions for the owner and read-only for the group and others

chmod 755 script.sh # Sets read, write, and execute for the owner, and read and execute for the group and others

Default Permissions:

When you create a new file or directory, it inherits permissions from the parent directory. The default permissions can be modified using the umask command, which subtracts the specified value from the maximum permissions (777 for directories, 666 for files) to get the default permissions.

Understanding and managing file permissions is crucial for maintaining security and controlling access to files on a Linux system. Be cautious while changing permissions, as incorrect settings can lead to unintended consequences or security vulnerabilities.

3. Inode and inode structure in Linux

Introduction to inode:

Inodes are a fundamental concept in Unix-like file systems, including Linux. An inode (index node) is a data structure that stores metadata about a file, except its name and actual data content. Each file in the file system is associated with a unique inode, which contains crucial information about the file, such as its permissions, ownership, timestamps, size, and disk block locations.

Inode Structure:

The inode structure in Linux typically contains the following important fields:
File Type: Indicates whether the inode corresponds to a regular file, directory, symbolic link, device file, etc.
File Permissions: Specifies the access permissions for the file, such as read, write, and execute permissions for the owner, group, and others.
Ownership: Stores the user ID (UID) and group ID (GID) of the file owner and associated group.
Timestamps: Records the last access time, last modification time, and last status change time of the file.
File Size: Indicates the size of the file in bytes.
Number of Links: Represents the number of hard links pointing to the same inode (used for file reference counting).
Disk Block Pointers: Stores pointers to the data blocks on the disk that hold the file's content. For small files, the inode contains direct block pointers, while for larger files, it includes indirect and double indirect pointers to locate data blocks.

Associated Commands:

Several Linux commands allow you to interact with inodes and retrieve inode-related information:
ls -i: Lists files along with their associated inode numbers.
stat: Displays detailed information about a file, including inode number, permissions, timestamps, and more.

stat filename

find: Can be used to search for files based on various criteria, including inode number.

find /path/to/search -inum <inode_number>

df -i: Shows inode-related information for the file systems mounted on the system.
debugfs: A powerful command-line tool to interactively view and manipulate the file system's internal structures, including inodes. Use it with caution as it provides low-level access to the file system.

debugfs /dev/sdaX# Replace 'X' with the appropriate partition number.

Importance of inodes:

Inodes play a crucial role in the management of file systems. Some key benefits of using inodes include:
Efficient storage management: Inodes allow the file system to manage files of various sizes effectively and allocate disk blocks efficiently.
Multiple links: Inodes support the concept of hard links, enabling multiple directory entries to point to the same file content.
Fast file access: The use of inodes allows the file system to locate and retrieve file metadata and content quickly.

In summary, inodes are essential data structures in Linux file systems that hold crucial information about files. Understanding how inodes work can provide insights into file system performance and behavior, making it easier to manage and troubleshoot storage-related issues.

4. Links in Linux

Introduction to Links:

In Linux, links are used to create references to files and directories. Links allow multiple paths to access the same underlying data, enabling efficient data organization and management. There are two types of links in Linux: Hard Links and Symbolic Links.

1. Hard Links:

- A hard link is a direct link to the inode of a file. It shares the same inode number as the original file and points directly to the data blocks containing the file's content.
- Changes made to one hard link are reflected in all hard links that share the same inode. All hard links are essentially equivalent; there is no concept of a "source" or "target" hard link.
- Hard links cannot span across different file systems or partitions because they are based on the inode, which is specific to each file system.
- If the original file is deleted, the data remains accessible through the hard links as long as there is at least one hard link pointing to the inode.

Associated Commands for Hard Links:

Creating a hard link:

```
ln /path/to/originalfile /path/to/hardlink
```

Checking the number of hard links to a file:

```
ls -l /path/to/file
```

Deleting a hard link (does not affect the original file):

```
rm /path/to/hardlink
```

2. Symbolic Links (Soft Links):

- A symbolic link, or soft link, is a special file that contains the path to another file or directory. It references the target file by its path name, rather than using the inode directly.
- Symbolic links can span across different file systems or partitions because they use the file path instead of the inode.
- Deleting the original file will render the symbolic link broken, as it will point to a nonexistent target.

Associated Commands for Symbolic Links:

Creating a symbolic link:

```
ln -s /path/to/originalfile /path/to/symlink
```

Checking if a file is a symbolic link:

```
ls -l /path/to/file
```

Deleting a symbolic link (does not affect the target file):

```
rm /path/to/symlink
```

Tips and Best Practices:

- Hard links cannot reference directories; use symbolic links for linking directories.
- Avoid creating symbolic links with relative paths if the target may be moved, as it can break the link.
- When removing files with hard links, ensure to delete the last hard link to free the disk space fully.

Links in Linux provide powerful tools for managing file and directory references efficiently. Hard links allow multiple names to refer to the same file's data blocks, while symbolic links create references based on file paths. Understanding and correctly using links contribute to a well-organized and easily maintainable file system.

5. Inode Associated with Directories and Access Permissions

Inode and Directories:

In Linux, directories are also represented by inodes. An inode associated with a directory contains metadata about the directory, such as its permissions, ownership, timestamps, and a list of filenames along with their corresponding inode numbers. The directory's inode points to the data blocks that store the filenames and their respective inodes. This hierarchical structure allows the file system to organize and access files efficiently.

Access Permissions on Directories:

Access permissions on directories are distinct from those on regular files. Directory permissions control what actions users can perform within the directory, such as listing its contents, creating, renaming, or deleting files within it.

The three main permissions for directories are:

1. **Read (r):** If a user has read permission on a directory, they can view the list of files and subdirectories within the directory using commands like ls.
2. **Write (w):** Write permission on a directory allows users to create, delete, and rename files or directories within it. They can also modify the contents of files within the directory if they have the appropriate permissions on those files.
3. **Execute (x):** Execute permission on a directory is required to access its contents. Without execute permission, users won't be able to change their working directory to the directory or access files within it.

Effects of Access Permissions on Directories:

Read Permission: Users with read permission can see the filenames and subdirectory names within the directory. However, they cannot access files or subdirectories unless they have the appropriate permissions on those objects.
Write Permission: Users with write permission can add or delete files or directories within the directory. They can also rename files or directories within the directory if they have write permission on the parent directory.
Execute Permission: Execute permission is essential for accessing the contents of a directory. Without execute permission, users won't be able to use the directory's name to access its contents or change their working directory to it.

Setting Directory Permissions:

You can use the chmod command to set permissions on directories.

For example:

```
chmod 755 directory_name
```

The above command grants read, write, and execute permissions to the owner and read and execute permissions to the group and others.

Directory Access Permissions and Security:

Properly managing directory access permissions is crucial for maintaining the security and privacy of your files. Setting appropriate permissions ensures that only authorized users can access, modify, or delete files within the directory. Avoid granting unnecessary permissions to prevent unauthorized access.
Understanding the relationship between inodes and directories, as well as the significance of directory access permissions, helps in effectively organizing and securing data within a Linux file system.

File and Directory Commands

1. File and Directory Commands

Linux Commands for Navigating and Listing Files in the Directory

cd - Change Directory:

The cd command is used to change the current working directory in the Linux terminal. It allows you to navigate to a different directory. If you omit any path, cd will take you to your home directory.

Example:

```
cd /path/to/directory
cd ..  # Move up one level in the directory hierarchy
cd ~   # Go to the home directory
```

pwd - Print Working Directory:

The pwd command prints the full path of the current working directory.

Example:

```
pwd
```

ls - List Files and Directories:

The ls command is used to list files and directories in the current working directory. By default, it shows only the names of visible files and directories.

Examples:

```
ls
ls /path/to/directory
```

ls -l - Long Format Listing:

The ls -l command provides a detailed listing of files and directories in the long format. It includes additional information like permissions, owner, group, size, modification date, and filename.

Example:

```
ls -l
```

ls -a - List Hidden Files:

The ls -a command shows all files, including hidden files (files whose names start with a dot .). Hidden files are usually configuration files or directories that are not displayed by default.

Example:

```
ls -a
```

ls -h - Human-Readable File Sizes:

The ls -h command displays file sizes in a human-readable format, using units like KB, MB, GB, etc., instead of just displaying bytes.

Example:

```
ls -lh
```

ls -t - Sort by Modification Time:

The ls -t command sorts files and directories based on their modification time, with the newest files shown first.

Example:

```
ls -lt
```

ls -R - Recursive Listing:

The ls -R command lists files and directories recursively, showing the content of subdirectories as well.

Example:

```
ls -R
```

These Linux commands for navigating and listing files in the directory provide essential tools for exploring the file system, managing files, and finding the information you need efficiently. Combining these commands with different options allows you to tailor the output according to your requirements.

2. Linux Commands for Creating and Deleting Files and Directories

touch - Create Empty Files or Update Timestamps:

The touch command is used to create an empty file or update the timestamps (access and modification) of an existing file. If the file already exists, touch will update its timestamps to the current time.

Examples:

```
touch filename      # Create an empty file named 'filename'
touch file1 file2   # Create multiple empty files
```

```
touch -a file.txt    # Update only the access time
touch -m file.txt    # Update only the modification time
```

mkdir - Create Directories:

The mkdir command is used to create directories (folders) in the file system.

Examples:

```
mkdir dirname              # Create a directory named 'dirname'
mkdir -p /path/to/directory# Create parent directories if they don't exist
```

rm - Remove (Delete) Files or Directories:

The rm command is used to remove (delete) files or directories. By default, it does not delete directories, but you can use the -r option (recursive) to remove directories and their contents.

Examples:

```
rm filename            # Remove a file named 'filename'
rm file1 file2         # Remove multiple files
rm -r dirname          # Remove a directory and its contents recursively
rm -rf /path/to/directory  # Forcefully remove a directory and its contents
```

rmdir - Remove Empty Directories:

The rmdir command is used to remove empty directories from the file system. It cannot remove directories with contents.

Example:

```
rmdir empty_directory# Remove an empty directory named 'empty_directory'
```

3. Linux Commands for Copying and Moving Files and Directories

cp - Copy Files and Directories:

The cp command is used to copy files or directories from one location to another. It can copy single or multiple files and directories. If copying directories, the -r (recursive) option is required to copy the directory and its contents.

Examples:

```
cp file.txt /path/to/destination       # Copy a file to a new location
cp file1.txt file2.txt /path/to/destination  # Copy multiple files to a directory
cp -r directory /path/to/destination  # Copy a directory and its contents to a new location
```

mv - Move (Rename) Files and Directories:

The mv command is used to move (rename) files or directories. If the source and destination are on the same filesystem, mv will perform a move operation. Otherwise, it will perform a rename. This command is also used to rename files or directories.

Examples:

```
mv oldname.txt newname.txt          # Rename a file
mv file.txt /path/to/directory      # Move a file to a different location
mv file1.txt file2.txt /path/to/directory  # Move multiple files to a directory
```

These Linux commands provide various options for copying and moving files and directories, making it easy to organize and transfer data efficiently within a file system or between systems.

4. Linux Commands for Permissions and Ownership of a File or Directory

chmod - Change File Permissions:

The chmod command is used to change the permissions of a file or directory. It allows you to modify read (r), write (w), and execute (x) permissions for the owner, group, and others.

Symbolic notation: You can use symbols (+, −, =) to add, remove, or set permissions.

For example:

```
chmod +x script.sh   # Adds execute permission for all (owner, group, others)
chmod u+w file.txt   # Adds write permission for the owner
chmod go-r file.txt  # Removes read permission for the group and others
chmod o=rw file.txt  # Sets read and write permissions for others
```

Octal notation: You can use numerical representation (0–7) to specify permissions directly.

For example:

```
chmod 644 example.txt  # Sets read and write permissions for the owner and read-only for the group and others
chmod 755 script.sh    # Sets read, write, and execute for the owner, and read and execute for the group and others
```

chown - Change File Ownership:

The chown command is used to change the owner and/or group ownership of a file or directory. You can specify the new owner and group using the user and group names or their corresponding user IDs (UID) and group IDs (GID).

Examples:

```
chown user:group file.txt      # Change the owner and group of the file
chown user file.txt            # Change only the owner of the file
chown :group file.txt          # Change only the group of the file
```

```
chown user: file.txt          # Remove the group ownership (set group to default)
```

chgrp - Change Group Ownership:

The chgrp command is used to change the group ownership of a file or directory. You can specify the new group using the group name or its corresponding GID.

Example:

```
chgrp group file.txt          # Change the group of the file
```

ls with -l Option - Display Permissions and Ownership:

The ls command with the -l option provides a detailed listing that includes permissions, ownership, timestamps, and more for files and directories.

Example:

```
ls -l
```

id - Display User and Group Information:

The id command displays the current user's UID, GID, and group membership. It is useful to identify the current user's ownership when dealing with file permissions.

Example:

```
id
```

Note: Changing permissions and ownership should be done with caution, as it can affect file access and security. Always ensure that you have the necessary permissions to perform such operations and carefully verify your changes before applying them, especially when altering system files or directories.

These Linux commands provide powerful tools for managing file permissions and ownership, allowing you to control access and maintain security within your file system.

5. Linux Commands for Disk Usage and Links

du - Disk Usage:

The du command is used to estimate and display the disk usage of files and directories in the file system. It helps you identify which directories or files consume the most disk space.

Options:

- -h: Displays sizes in human-readable format (e.g., KB, MB, GB).
- -s: Displays only the total size of each argument (directory).
- -c: Includes a grand total of disk usage at the end of the output.

Examples:

```
du -h                  # Displays disk usage of current directory and its subdirectories
du -h /path/to/directory  # Displays disk usage of a specific directory
du -sh /path/to/directory # Displays only the total disk usage of the directory
du -h --max-depth=1       # Displays disk usage of immediate subdirectories only
du -ch /path/to/directory # Displays the total disk usage, including the grand total
```

df - Disk Free:

The df command shows the amount of free and used disk space on mounted file systems. It provides an overview of the space available on each file system.

Options:

-h: Displays sizes in human-readable format (e.g., KB, MB, GB).

-T: Displays the file system type along with disk usage.

Examples:

```
df -h           # Displays disk space information for all mounted file systems
df -h /dev/sda1  # Displays disk space information for a specific file system
df -hT          # Displays disk space information with file system types
```

ln - Create Links:

The ln command is used to create hard links and symbolic links.

Options:

-s: Creates a symbolic link instead of a hard link.

Examples:

```
ln /path/to/originalfile /path/to/hardlink      # Creates a hard link
ln -s /path/to/originalfile /path/to/symlink   # Creates a symbolic link
```

readlink - Display Symbolic Link Value:

The readlink command is used to display the value of a symbolic link.

Example:

```
readlink /path/to/symlink   # Display the target of the symbolic link
```

unlink - Remove Links:

The unlink command is used to remove (delete) links. It can be used to remove both hard links and symbolic links.

Example:

```
unlink /path/to/link   # Remove the link (whether hard link or symbolic link)
```

These Linux commands provide valuable insights into disk usage, allowing you to monitor storage capacity and identify potential space hogs. The ability to create and manage links efficiently helps organize files and directories and improve file system structure and accessibility.

6. Additional Linux Commands

1. grep - Search Text:

The grep command is used for pattern matching and searching text in files or standard input. It can search for a specific string or regular expression within files and display matching lines.
Example:

```
grep "pattern" filename
```

2. find - Search for Files and Directories:

The find command is used to search for files and directories based on various criteria such as name, size, type, and modification time. It is a powerful tool for locating specific files in a directory tree.
Example:

```
find /path/to/directory -name "*.txt"
```

3. top - Monitor System Activity:

The top command displays real-time information about system activity, including CPU usage, memory usage, running processes, and more. It is useful for monitoring system performance and identifying resource-hungry processes.
Example:

```
top
```

4. ps - Display Process Status:

The ps command shows information about running processes on the system. It provides details such as process ID (PID), CPU and memory usage, parent process ID (PPID), and more.
Example:

```
ps aux
```

5. kill - Terminate Processes:

The kill command is used to send signals to processes, allowing you to terminate or control them. The default signal is SIGTERM, which terminates a process gracefully.
Example:

```
kill PID
```

6. history - Display Command History:

The history command shows a list of previously executed commands in the current session. It is useful for recalling and reusing commands without having to type them again.

Example:

history

7. man - Display Manual Pages:

The man command is used to display the manual pages (documentation) for various commands and system functions. It provides detailed information about command usage, options, and examples.

Example:

man ls

These additional Linux commands expand your control over the system, enabling you to search for text, monitor processes, manage running programs, and access useful documentation. They further enhance your experience and efficiency when working with Linux systems.