Linux Shell & Server

UNIT - I

Introduction to Linux

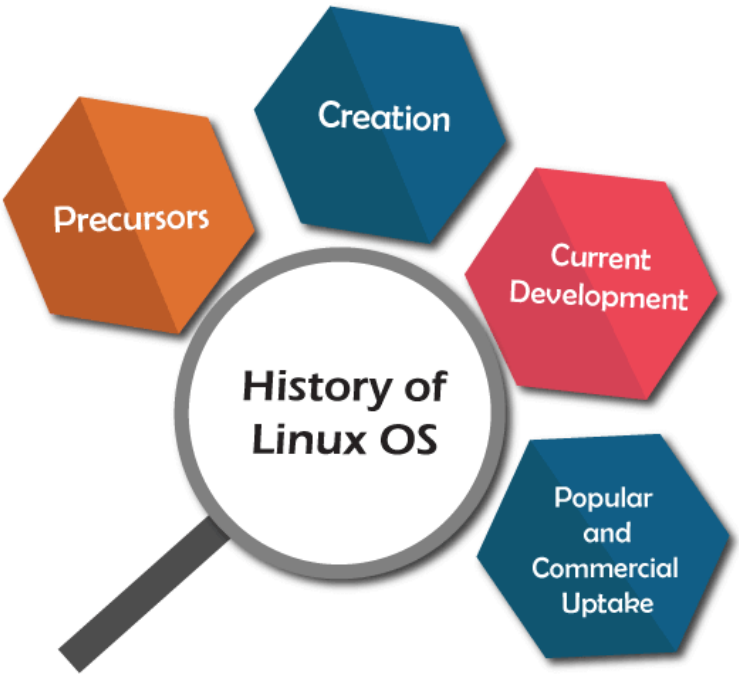
Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by ***Linus Torvalds***. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term ***"Linux"*** in the title, but the Free Software Foundation uses the ***"GNU/Linux"*** title to focus on the necessity of GNU software, causing a few controversies.

Famous Linux distributions are Ubuntu, Fedora Linux, and Debian, the latter of which is composed of several different modifications and distributions, including Xubuntu and Lubuntu. Commercial distributions are SUSE Linux Enterprise and Red Hat Enterprise Linux. Desktop distributions of Linux are windowing systems like Wayland or X11 and desktop environments like KDE Plasma and GNOME.

- Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems.
- Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022.
- However, Linux is used by just around 2.6% of desktop computers as of November 2022.
- Also, Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system.
- It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers.

Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License). For example, the Linux kernel is licensed upon the GPLv2.

History of Linux OS



- **Precursors**
  The Unix-based operating system was implemented and conceived in 1969 at ***AT&T's Bell*** labs by Joe Ossanna, Douglas McIlroy, Dennis Ritchie, and Ken Thompson in the United States. First published in 1971, Unix was entirely written in assembly language, as was the basic practice at the time. It was updated in the C language by Dennis Ritchie in a key pioneering way in 1973. The availability of a Unix high-level language implementation made its porting to distinct computer platforms convenient.

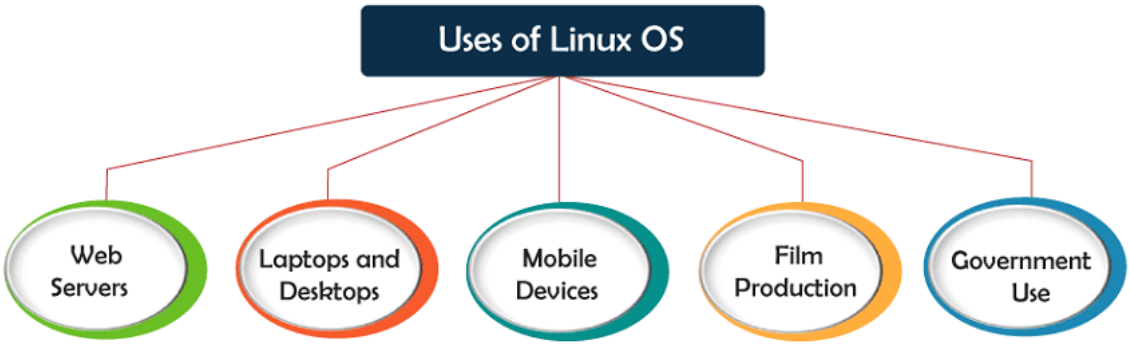
- **Creation**
  Torvalds registered in a Unix course while visiting the University of Helsinki in the 1990's fall. The course used a MicroVAX minicomputer executing Ultrix, and one of the needed texts was Operating Systems: Design and Implementation by Andrews S. Tanenbaum. The textbook contained a copy of the MINIX operating system of Tanenbaum. It was together with this course that Torvalds initially became open to Unix. He became interested in operating systems in 1991. Frustrated by the MINIX licensing, which limited it to only educational use at the time, he started to work on his operating system kernel, which became the Linux Kernel eventually.
  Torvalds started the Linux kernel development on MINIX, and software written for MINIX was used on Linux as well. Later, Linux was cultivated, and then the development of the Linux kernel appeared on Linux systems. Also, GNU applications replaced every MINIX component due to it was beneficial to use the free code through the GNU Project with the new OS; code licensed upon the GNU GPL can be re-applied in other computer functions as long as they are also published under a compatible or the same license.
  Torvalds started a switch from his actual license, which banned commercial distribution, to the GNU GPL. Developers operated to develop GNU elements with the Linux Kernel, making a free and fully functional operating system.

- **Current development**
  The lead maintainer of the Linux Kernel is ***Greg Kroah-Hartman*** who guides its development. The e executive director for the Free Software Foundation is ***William John Sullivan***, which in turn supported the GNU components. Corporations and individuals finally develop non-GNU third-party components.
  The third-party components are composed of a wide body of work and may contain both user libraries and applications, and kernel modules. Linux community and vendors distribute and combine the kernel, non-GNU components, and GNU components with extra package management software in the fashion of Linux distributions.

- **Popular and commercial uptake**
  In production environments, Linux adoption began to take off initially in the mid-1990s in the supercomputing community instead of being used by only hobbyists, where organizations like NASA began to increasingly replace their expensive machines with inexpensive commodity computer clusters running Linux. Commercial use started when IBM and Dell, pursued by Hewlett-Packard, began providing Linux support for escaping the monopoly of Microsoft in the desktop OS market.
  Linux systems are completely used in computing today, from embedded systems to every supercomputer virtually, and have secured a position in server installations like the famous LAMP application stack. The usage of Linux distributions in enterprise and home desktops has been developing.
  Also, Linux distributions have become famous in the netbook market, with several devices moving with installed customized Linux distributions and Google publishing their ChromeOS developed for netbooks.

Uses of Linux OS

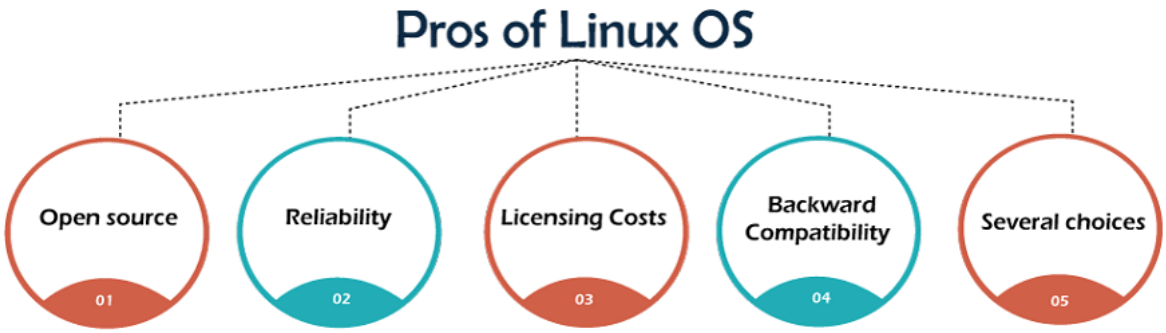
Several quantitative studies of open-source/free software concentrate on topics, such as reliability and market share, with many studies examining Linux specifically. The Linux market is developing, and the Linux OS market size is supposed to see a development of 19.2% by 2027, reaching 15.64 billion dollars, compared to 3.89 billion in 2019. Proponents and analysts attribute the associative Linux success to its freedom, low cost, reliability, and security from vendor lock-in.



- **Web servers**
  W3Cook releases stats that utilize the top 1,000,000 Alexa domains, which estimate that 96.55% of web servers use Linux, 1.73% use Windows, and 1.72% use FreeBSD as of May 2015.
- **Laptops and desktops**
  As of May 2022, the estimated Linux market share is around 2.5% on desktop computers, according to web server statistics. Microsoft Windows include a market share of approximately 75.5%, while macOS has around 14.9%.
- **Mobile devices**
  Android has become the leading OS for smartphones which is Linux kernel-based. In July 2022, 71.9% of smartphones worldwide using the internet used Android. Also, Android is a famous OS for tablets, being liable for more than 60% of table sales as of 2013.
- **Film production**
  Linux has been the preferred platform in the film industry for years. The first big film released on a Linux server was 1997's Titanic. Since then, big studios, including Industrial Light & Magic, Weta Digital, Pixar, and DreamWorks Animation, have relocated to Linux.
- **Government use**
  Linux distros have also got popularity in several national and local governments. Kerala has gone to the mandating extent that every state high school use Linux on their systems. China utilizes Linux exclusively as the OS for its Loongson processor family for achieving technology independence.
  A few regions have integrated their Linux distribution in Spain, which is extensively used in official and educational institutions. Also, Germany and France have taken steps toward Linux adoption. The Red Star OS of North Korea is based on a Fedora Linux version developed since 2002.

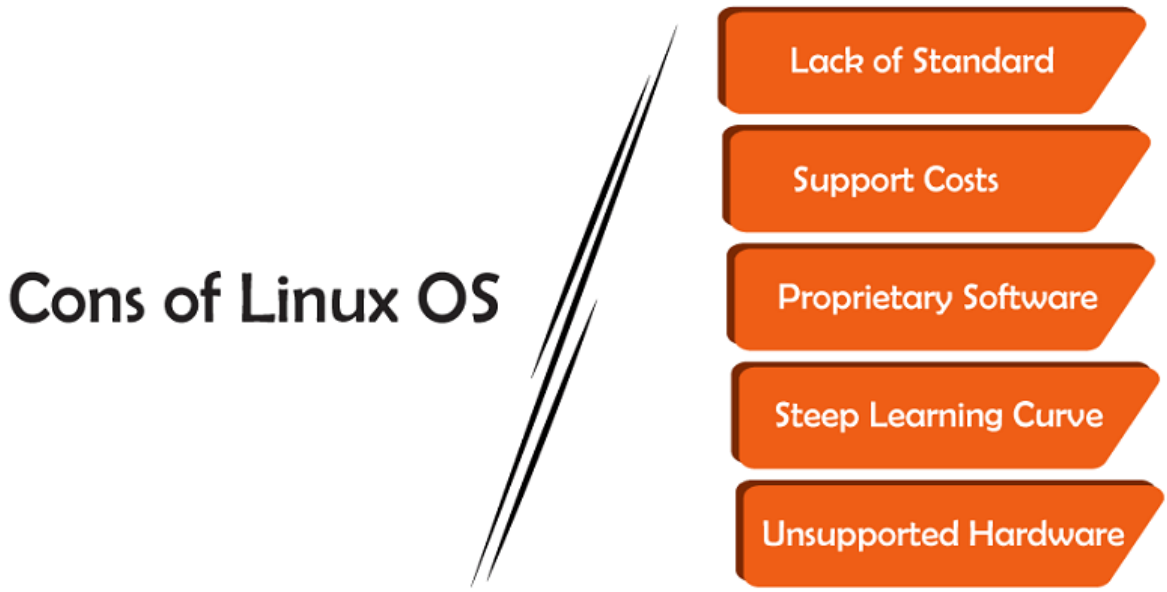
Pros and Cons of Linux OS

Some benefits of using Linux are listed and explained below:



- **Open source:** The Linux kernel is published under the open-source software license of GNU GPL. Most distributions contain several applications with various options in almost all categories. Also, several distributions contain proprietary software, like device drivers offered by manufacturers, to support hardware.
- **Reliability:** Linux is treated as a reliable operating system, and it is well-supported with several security patches. Also, Linux is treated as a stable OS, which means it can execute in almost every circumstance. Linux can also handle errors when running unexpected input and software.
- **Licensing costs:** Linux has no accurate licensing fees, unlike Apple macOS or Microsoft Windows. While system support is present for a fee from several Linux vendors, the operating system itself is free to use and copy. A few IT organizations have enhanced their savings by moving their server software to Linux from a commercial operating system.
- **Backward compatibility:** Linux and many open-source software tend to be frequently updated for functional and security patches while having core functionality. Shell scripts and configurations are likely to operate unchanged even if software updates are used. Generally, Linux and other open-source applications do not alter their operation modes with new versions, unlike economic software vendors that mount new releases of their operating systems with new forms of work.
- **Several choices:** Between almost all infinite options, several available distros, and many application options to configure, compile, and run Linux on almost all hardware platforms, it's possible to develop Linux for almost all applications.

A few drawbacks of Linux are:



- **Lack of standard:** No standard version is available for Linux, which may be nice to optimize Linux for specific applications, but less so to deploy desktop images and standardized servers. The huge variety of options can convolute support as an outcome.
- **Support costs:** Support isn't free, while an organization can freely acquire Linux without licensing fees. Almost all enterprise Linux distributors, such as Red Hat and SUSE, provide support contracts. These license fees can significantly decrease savings depending on the situation.
- **Proprietary software:** PC productivity software, such as Microsoft Office, can't be utilized on Linux desktops, and many proprietary software may not be available for Linux platforms.
- **Steep learning curve:** Several users battle to learn to use Linux-based applications and Linux desktops.
- **Unsupported hardware:** Several hardware manufacturers enable the device drivers of Linux accessible for their products, but several don't.

Linux Basic Features

Linux is a powerful and flexible operating system that offers a range of features, making it suitable for a variety of environments, from servers and desktops to embedded systems. Here are some of the basic features that make Linux popular:

1. Open Source

- **Free Software**: Linux is open source, meaning its source code is freely available for anyone to view, modify, and distribute. This encourages collaboration and innovation.
- **Community Development**: Linux is developed and maintained by a global community of contributors, allowing for rapid updates and improvements.

2. Multitasking

- **Concurrent Processes**: Linux can handle multiple tasks simultaneously, allowing users to run several applications at the same time without performance degradation.
- **Process Management**: Linux provides robust process management tools to monitor and control running applications and services.

3. Multi-User

- **Multiple User Accounts**: Linux supports multiple user accounts, allowing multiple users to work on the same system without interfering with each other.
- **User Permissions**: Access to files and resources can be controlled through a robust permissions system, enhancing security and privacy.

4. Security

- **Permissions and Ownership**: Linux uses a permission-based system to control access to files and directories, with settings for the owner, group, and others.
- **Firewalls**: Tools like iptables and firewalld allow users to configure network security policies.
- **SELinux and AppArmor**: Provide additional layers of security by enforcing mandatory access controls.

5. Portability

- **Cross-Platform Compatibility**: Linux runs on a wide range of hardware platforms, from desktops and servers to smartphones and embedded devices.
- **Minimal Hardware Requirements**: Linux can run efficiently on older and less powerful hardware, making it suitable for various environments.

6. Stability and Reliability

- **Robust Performance**: Linux is known for its stability and reliability, often running for long periods without the need for reboots.
- **Crash Recovery**: Advanced file systems and recovery tools help maintain data integrity and recover from crashes.

7. Networking

- **Built-In Networking Tools**: Linux includes comprehensive networking support, with tools for managing network connections and services.
- **Server Capabilities**: Linux is widely used for hosting web, email, and file servers, thanks to its robust networking stack.

8. Shell and Command Line Interface

- **Powerful Shells**: Linux provides several powerful command-line shells, such as Bash, Zsh, and Fish, for task automation and scripting.
- **Scripting**: Users can write scripts to automate repetitive tasks and enhance productivity.

9. Package Management

- **Software Repositories**: Linux distributions use package managers (e.g., APT, YUM, DNF) to install, update, and manage software packages.
- **Dependency Management**: Package managers handle software dependencies, ensuring that all necessary components are installed.

10. Customization

- **Desktop Environments**: Users can choose from various desktop environments, such as GNOME, KDE, and XFCE, to customize their user interface.
- **Theming and Extensions**: Linux allows for extensive customization of appearance and functionality through themes and extensions.

11. Virtualization

- **Virtual Machines**: Linux supports virtualization technologies like KVM, Xen, and Docker, enabling users to run multiple operating systems on a single machine.
- **Containerization**: Tools like Docker and Kubernetes provide lightweight alternatives to full virtual machines, ideal for cloud applications.

Different Flavors of Linux

Linux comes in various distributions, often called "flavors" or "distros," each tailored for specific use cases, user preferences, or hardware requirements. These distributions package the Linux kernel with various software, tools, and configurations to suit different needs. Here are some of the most popular and notable Linux distributions:

1. Ubuntu

- **Base**: Debian
- **Target Audience**: Beginners, general users, and developers.
- **Features**:
    - User-friendly interface.
    - Regular releases every six months with Long Term Support (LTS) versions every two years.
    - Extensive community support and documentation.
- **Desktop Environment**: GNOME (default), with flavors like Kubuntu (KDE), Xubuntu (XFCE), and Lubuntu (LXQt).

2. Debian

- **Base**: Independent
- **Target Audience**: Experienced users and server environments.
- **Features**:
    - Stability and reliability with a focus on free software.
    - A vast repository of packages.
    - Long release cycles ensure stability.
- **Desktop Environment**: GNOME (default), but highly customizable.

3. Fedora

- **Base**: Red Hat
- **Target Audience**: Developers and bleeding-edge users.
- **Features**:
    - Latest software and technologies with frequent updates.
    - Strong focus on open-source and free software.
    - Used as a testing ground for Red Hat Enterprise Linux (RHEL).
- **Desktop Environment**: GNOME (default).

4. CentOS (and CentOS Stream)

- **Base**: Red Hat
- **Target Audience**: Enterprise users and servers.
- **Features**:
    - Binary-compatible with Red Hat Enterprise Linux (RHEL).
    - Known for stability and reliability in enterprise environments.
    - CentOS Stream is a rolling-release version.
- **Desktop Environment**: GNOME (default).

5. Red Hat Enterprise Linux (RHEL)

- **Base**: Independent
- **Target Audience**: Enterprises and businesses.

- **Features**:
    - Commercial support and services.
    - Certified for various hardware and software environments.
    - Long-term support and stability.
- **Desktop Environment**: GNOME (default).

6. Arch Linux

- **Base**: Independent
- **Target Audience**: Advanced users and enthusiasts.
- **Features**:
    - Rolling-release model ensures the latest software updates.
    - Highly customizable and minimalistic.
    - Extensive documentation through the Arch Wiki.
- **Desktop Environment**: None by default; users choose and configure their own.

7. Manjaro

- **Base**: Arch Linux
- **Target Audience**: Intermediate users who want Arch's power with ease of use.
- **Features**:
    - User-friendly installation and configuration.
    - Pre-configured desktop environments.
    - Access to Arch repositories and user-friendly tools.
- **Desktop Environment**: XFCE (default), KDE, GNOME, and others.

8. openSUSE

- **Base**: Independent
- **Target Audience**: Developers, system administrators, and general users.
- **Features**:
    - Two versions: openSUSE Leap (stable) and Tumbleweed (rolling release).
    - Comprehensive configuration tool (YaST).
    - Strong community support.
- **Desktop Environment**: KDE Plasma (default) and GNOME.

9. Linux Mint

- **Base**: Ubuntu/Debian
- **Target Audience**: Beginners and desktop users.
- **Features**:
    - User-friendly with a focus on out-of-the-box usability.
    - Various desktop environments tailored for ease of use.

    - Cinnamon desktop environment, which is known for being intuitive.
- **Desktop Environment**: Cinnamon (default), MATE, and XFCE.

10. Kali Linux

- **Base**: Debian
- **Target Audience**: Security professionals and ethical hackers.
- **Features**:
    - Pre-installed with security and penetration testing tools.
    - Focus on security and forensic analysis.
    - Customizable and lightweight.
- **Desktop Environment**: XFCE (default), GNOME, and KDE.

11. Elementary OS

- **Base**: Ubuntu
- **Target Audience**: Desktop users seeking a polished and aesthetically pleasing interface.
- **Features**:
    - Elegant and intuitive interface inspired by macOS.
    - Focus on simplicity and ease of use.
    - Curated app store with high-quality applications.
- **Desktop Environment**: Pantheon.

12. Zorin OS

- **Base**: Ubuntu
- **Target Audience**: Beginners and users transitioning from Windows.
- **Features**:
    - Windows-like user interface.
    - User-friendly with built-in tools to ease transition from other operating systems.
    - Customization options and support for Windows applications.
- **Desktop Environment**: Zorin Desktop (based on GNOME).

13. Slackware

- **Base**: Independent
- **Target Audience**: Experienced users seeking simplicity and control.
- **Features**:
    - One of the oldest distributions, known for simplicity and minimalism.
    - Text-based installation and configuration.
    - Focus on stability and performance.
- **Desktop Environment**: KDE and XFCE.

14. Gentoo

- **Base**: Independent
- **Target Audience**: Advanced users and enthusiasts.
- **Features**:
    - Source-based distribution allowing optimization and customization.
    - Portage package management system for compiling from source.
    - Highly flexible and configurable.
- **Desktop Environment**: None by default; users choose and configure their own.

Installing Requirements of Linux

Installing Linux generally requires a few key components:

1. **Processor**: An Intel or AMD x86 processor (or ARM, depending on the distribution). Most modern processors are supported.
2. **RAM**:
    - **Minimum**: 512 MB (for lightweight distributions).
    - **Recommended**: 1 GB or more for a smoother experience, especially with more feature-rich distributions.
3. **Storage**:
    - **Minimum**: 5 GB of free space for a basic installation.
    - **Recommended**: 10 GB or more for better performance and more applications.
4. **Graphics**: Any graphics card that supports a screen resolution of at least 800x600. More recent distributions may benefit from more advanced graphics hardware.
5. **Boot Device**:
    - A USB flash drive or CD/DVD for the installation media.
    - Alternatively, you can use a network-based installation if supported by your distribution.
6. **Internet Connection**: Not required for installation, but helpful for downloading updates and additional software.
7. **Compatibility**: Ensure your hardware is compatible with the Linux distribution you plan to install. Popular distributions like Ubuntu or Fedora have extensive hardware support.
8. **BIOS/UEFI Settings**: You may need to adjust your BIOS/UEFI settings to boot from your installation media and configure the boot order accordingly.

Basic Architecture of Unix/Linux System

The Unix/Linux operating system architecture is a well-structured system that divides responsibilities among various components. Here's a detailed explanation of each part:

1. Kernel

- **Definition**: The kernel is the heart of the Unix/Linux operating system. It is responsible for managing the system's hardware and providing essential services to the software running on the system.
- **Functions**:
    - **Process Management**: Handles process creation, scheduling, and termination. It ensures that processes are allocated CPU time and resources.
    - **Memory Management**: Manages the system's memory, including virtual memory and physical memory. It allocates memory to processes and handles swapping between RAM and disk storage.
    - **File System Management**: Provides a framework for file operations, including reading, writing, and organizing files in a hierarchical directory structure.
    - **Device Management**: Interfaces with hardware devices through device drivers, allowing software to interact with hardware components like disks, printers, and network cards.
    - **System Calls**: Provides a set of functions that programs use to request services from the kernel, such as file operations and process control.

2. System Libraries

- **Definition**: System libraries are collections of pre-written code that applications use to perform common tasks. They provide a standardized interface to the kernel's functions.
- **Functions**:
    - **Abstraction**: Libraries abstract complex system calls into simpler functions that are easier for applications to use.
    - **Shared Code**: Libraries are shared among multiple applications, which helps reduce redundancy and save memory.
- **Example**: The GNU C Library (glibc) is a widely used library in Linux systems that provides standard C library functions.

3. System Utilities

- **Definition**: System utilities are essential tools and programs that perform various system-related tasks and operations.
- **Functions**:
    - **File Management**: Tools like ls, cp, mv, and rm help manage files and directories.
    - **System Monitoring**: Commands such as top, ps, and df provide information about system performance, running processes, and disk usage.
    - **Networking**: Utilities like ping, netstat, and ifconfig are used for network configuration and diagnostics.

4. Shell

- **Definition**: The shell is a command-line interface that allows users to interact with the operating system by entering commands.
- **Types**:
    - **Bourne Again Shell (bash)**: A widely used shell that provides features like command history and tab completion.
    - **Bourne Shell (sh)**: The original shell, known for its simplicity and portability.

    - **C Shell (csh)**: Known for its C-like syntax and job control features.
    - **Z Shell (zsh)**: Combines features from other shells and adds its own, such as improved completion and globbing.
- **Functions**:
    - **Command Execution**: Interprets and executes user commands.
    - **Scripting**: Allows users to write and execute shell scripts for automating tasks.

5. File System

- **Definition**: The file system organizes and manages data stored on storage devices. It provides a hierarchical structure for files and directories.
- **Functions**:
    - **Data Storage**: Stores files and directories on disk drives.
    - **Access Control**: Manages permissions and access rights for files and directories.
    - **File Management**: Provides operations like creating, deleting, and modifying files.
- **Types**:
    - **Ext4**: A widely used Linux file system known for its stability and performance.
    - **XFS**: A high-performance file system suited for large files and high-capacity storage.
    - **Btrfs**: A newer file system with features like snapshots and built-in RAID support.

6. User Interface

- **Definition**: The user interface allows users to interact with the operating system and applications.
- **Types**:
    - **Command-Line Interface (CLI)**: Provides a text-based interface where users type commands to perform tasks.
    - **Graphical User Interface (GUI)**: Provides a visual interface with windows, icons, and menus. Common Linux desktop environments include GNOME, KDE Plasma, and XFCE.

7. Applications

- **Definition**: Applications are software programs that run on the operating system and provide various functionalities to users.
- **Types**:
    - **Desktop Applications**: Include web browsers, text editors, and media players.
    - **Server Applications**: Include web servers, database servers, and file servers.

8. Device Drivers

- **Definition**: Device drivers are specialized programs that allow the kernel to communicate with hardware devices.
- **Functions**:
    - **Hardware Abstraction**: Provides a consistent interface for the kernel to interact with different hardware components.

- **Device Control**: Handles specific functions and operations of hardware devices, such as reading from or writing to storage devices.

9. Configuration Files

- **Definition**: Configuration files store settings and parameters that control the behavior of the operating system and applications.
- **Location**:
  - **System Configuration**: Typically located in /etc (e.g., /etc/fstab for file system mounts).
  - **User Configuration**: Typically located in user-specific directories (e.g., ~/.bashrc for shell settings).

linux standard directories

Linux follows a standard directory structure defined by the Filesystem Hierarchy Standard (FHS). This standard organizes the file system into a hierarchical directory tree. Here are some of the key standard directories you'll find in a typical Linux system:

1. / (Root)

- **Description**: The top-level directory in the hierarchy. All other directories are subdirectories of this directory.
- **Contents**: Contains all other directories and files.

2. /bin

- **Description**: Contains essential user binaries (executable programs) required for system boot and basic system operation.
- **Contents**: Common commands like ls, cp, mv, and cat.

3. /boot

- **Description**: Contains files needed for the boot process, including the Linux kernel and bootloader configuration files.
- **Contents**: Files such as vmlinuz (the kernel), initrd (initial RAM disk), and bootloader configuration files.

4. /dev

- **Description**: Contains device files that represent hardware devices or virtual devices.
- **Contents**: Special files for devices like /dev/sda (hard drive), /dev/tty (terminal), and /dev/null (null device).

5. /etc

- **Description**: Contains system-wide configuration files and scripts.
- **Contents**: Configuration files such as /etc/passwd (user account information), /etc/fstab (file system table), and /etc/hostname (system hostname).

6. /home

- **Description**: Contains user home directories. Each user has a subdirectory within /home where their personal files and settings are stored.
- **Contents**: User directories like /home/user1, /home/user2.

7. /lib

- **Description**: Contains shared library files required by the system and applications to run.
- **Contents**: Essential libraries used by binaries in /bin and /sbin, such as libc.so.

8. /media

- **Description**: Provides mount points for removable media such as USB drives, CDs, and DVDs.
- **Contents**: Mount points like /media/usb, /media/cdrom.

9. /mnt

- **Description**: A directory for temporarily mounting file systems.
- **Contents**: Typically empty but used for mounting external file systems or partitions temporarily, such as /mnt/data.

10. /opt

- **Description**: Contains optional application software packages and add-on applications.
- **Contents**: Directory for third-party applications, e.g., /opt/softwares.

11. /proc

- **Description**: A virtual filesystem providing process and kernel information.
- **Contents**: Dynamic information about system processes and hardware, such as /proc/cpuinfo and /proc/meminfo.

12. /root

- **Description**: The home directory for the root user (superuser).
- **Contents**: Files and configuration specific to the root user.

13. /run

- **Description**: Stores runtime information, such as process IDs and system state information.
- **Contents**: Files related to system runtime, such as /run/lock (lock files) and /run/shm (shared memory).

14. /sbin

- **Description**: Contains system binaries that are primarily used by the system administrator for system maintenance tasks.
- **Contents**: Essential system commands like fsck, reboot, and shutdown.

15. /srv

- **Description**: Contains data for services provided by the system. It is used for service-specific data.
- **Contents**: Data for services like web servers, e.g., /srv/www for web content.

16. /tmp

- **Description**: Contains temporary files created by system and application processes.
- **Contents**: Temporary files that can be deleted on reboot or by the system, e.g., /tmp/session_data.

17. /usr

- **Description**: Contains user-related programs and data.
- **Subdirectories**:
    - /usr/bin: User binaries.
    - /usr/lib: Shared libraries for user applications.
    - /usr/share: Architecture-independent data, such as documentation and localization files.

18. /var

- **Description**: Contains variable data files, such as logs, databases, and mail spools.
- **Contents**: Log files (/var/log), mail spools (/var/mail), and temporary files (/var/tmp).

Command for Files and Directories

1. cd (Change Directory)

- **Purpose**: Changes the current working directory.
- **Usage**: cd [directory]
- **Examples**:
    - cd /home/user/Documents – Changes to the /home/user/Documents directory.
    - cd .. – Moves up one level in the directory hierarchy.
    - cd ~ – Moves to the user's home directory.

2. ls (List)

- **Purpose**: Lists files and directories within the current directory.
- **Usage**: ls [options] [directory]
- **Options**:
    - -l – Long format, showing detailed information (permissions, owner, size, modification date).
    - -a – Includes hidden files (files starting with .).
    - -h – Human-readable sizes (e.g., KB, MB).
- **Examples**:
    - ls – Lists files in the current directory.
    - ls -la – Lists all files, including hidden ones, in long format.

3. cp (Copy)

- **Purpose**: Copies files or directories from one location to another.
- **Usage**: cp [options] source destination
- **Options**:
    - -r – Recursively copy directories.
    - -i – Prompts before overwriting files.
    - -u – Only copies files that are newer than the destination files.

- **Examples**:
    - cp file1.txt file2.txt – Copies file1.txt to file2.txt.
    - cp -r dir1/ dir2/ – Recursively copies dir1 to dir2.

4. mv (Move)

- **Purpose**: Moves or renames files and directories.
- **Usage**: mv [options] source destination
- **Options**:
    - -i – Prompts before overwriting.
    - -u – Only moves files that are newer than the destination files.
- **Examples**:
    - mv file1.txt file2.txt – Renames file1.txt to file2.txt.
    - mv file1.txt /home/user/ – Moves file1.txt to /home/user/.

5. rm (Remove)

- **Purpose**: Deletes files or directories.
- **Usage**: rm [options] file
- **Options**:
    - -r – Recursively remove directories and their contents.
    - -i – Prompts before each removal.
    - -f – Forces removal without prompts.
- **Examples**:
    - rm file1.txt – Deletes file1.txt.
    - rm -r dir1/ – Recursively deletes the dir1 directory and its contents.

6. mkdir (Make Directory)

- **Purpose**: Creates a new directory.
- **Usage**: mkdir [options] directory
- **Options**:
    - -p – Creates parent directories as needed.
- **Examples**:
    - mkdir newdir – Creates a directory named newdir.
    - mkdir -p parentdir/childdir – Creates parentdir and childdir if they do not exist.

7. rmdir (Remove Directory)

- **Purpose**: Deletes empty directories.
- **Usage**: rmdir directory
- **Examples**:
    - rmdir olddir – Deletes the empty directory olddir.

8. pwd (Print Working Directory)

- **Purpose**: Displays the path of the current working directory.
- **Usage**: pwd
- **Examples**:
    - pwd – Prints the absolute path of the current directory, e.g., /home/user/Documents.

9. file (File Type)

- **Purpose**: Determines the type of a file.
- **Usage**: file [file]
- **Examples**:
    - file example.txt – Displays the type of example.txt (e.g., ASCII text, executable).
    - file /bin/ls – Identifies the type of the binary executable file.

10. more

- **Purpose**: Displays the contents of a file one page at a time.
- **Usage**: more [file]
- **Examples**:
    - more largefile.txt – Shows the contents of largefile.txt with pagination.
- **Navigation**:
    - Press Space to move to the next page.
    - Press Enter to move one line at a time.
    - Press q to quit.

11. less

- **Purpose**: Similar to more, but with more advanced features for viewing file contents.
- **Usage**: less [file]
- **Examples**:
    - less largefile.txt – Opens largefile.txt with more flexible navigation than more.
- **Navigation**:
    - Press Space to move to the next page.
    - Press b to move back a page.
    - Press / to search forward.
    - Press ? to search backward.
    - Press q to quit.

Creating and Viewing files using cat

The cat command in Linux is a versatile tool used for creating, viewing, and concatenating files. Here's how you can use cat for these purposes:

Creating Files Using cat

1. Creating a New File

You can create a new file and add content to it using cat with the redirection operator >.

Command:

```
cat > filename
```

Example:

```
cat > myfile.txt
```

After entering the above command, you can type the content you want to add to the file. Press Ctrl+D to save and exit.

Steps:

1. Type cat > myfile.txt and press Enter.
2. Type the content of the file.
3. Press Ctrl+D to save the file and return to the command prompt.

2. Appending Content to an Existing File

To append content to an existing file, use cat with the append redirection operator >>.

Command:

```
cat >> filename
```

Example:

```
cat >> myfile.txt
```

After entering the command, you can type the content you want to append to the file. Press Ctrl+D to save and exit.

Steps:

1. Type cat >> myfile.txt and press Enter.
2. Type the content you want to append.
3. Press Ctrl+D to save the changes and return to the command prompt.

Viewing Files Using cat

The cat command can also be used to view the contents of one or more files.

1. Viewing a Single File

Command:

```
cat filename
```

Example:

```
cat myfile.txt
```

This will display the content of myfile.txt on the terminal.

2. Viewing Multiple Files

You can concatenate and view multiple files by specifying them in the command.

Command:

```
cat file1 file2
```

Example:

```
cat myfile.txt anotherfile.txt
```

This will display the contents of both myfile.txt and anotherfile.txt one after the other.

Practical Examples

Example 1: Creating a New File

```
$ cat > example.txt
This is a new file.
It contains several lines of text.
Ctrl+D
```

Output:

```
# (returns to the command prompt after pressing Ctrl+D)
```

Example 2: Viewing a File

```
$ cat example.txt
This is a new file.
It contains several lines of text.
```

Example 3: Appending to a File

```
$ cat >> example.txt
This line will be appended.
Ctrl+D
```

Output:

```
# (returns to the command prompt after pressing Ctrl+D)
```

Example 4: Viewing Multiple Files

```
$ cat example.txt anotherfile.txt
```

```
This is a new file.

It contains several lines of text.

This line will be appended.

# (Contents of anotherfile.txt follow)
```

The cat command is a powerful utility for file manipulation in Linux, allowing you to quickly create, append, and view file contents directly from the terminal.

File Comparisons - cmp&comm

File comparison is a common task in Linux, and there are various tools available for this purpose. Two commonly used commands for comparing files are cmp and comm. Here's a detailed explanation of each, including their usage and examples:

cmp (Compare)

The cmp command is used to compare two files byte by byte. It reports the first byte and line number where the files differ.

Syntax

```
cmp [options] file1 file2
```

Common Options

- -l: Outputs the byte numbers and the differing bytes.
- -s: Silent mode; reports nothing, but sets the exit status to indicate if the files are different.

Examples

1. **Basic Comparison**

```
cmp file1.txt file2.txt
```

If the files are identical, cmp returns nothing and exits with status 0. If they differ, it reports the byte and line number of the first difference.

2. **Show Byte Numbers and Differing Bytes**

```
cmp -l file1.txt file2.txt
```

This command lists all differing bytes with their positions.

3. **Silent Mode**

```
cmp -s file1.txt file2.txt
```

This command provides no output, but the exit status indicates if the files are identical (exit status 0) or different (exit status 1).

Example Output

For two different files:

```
cmp file1.txt file2.txt

file1.txt file2.txt differ: byte 5, line 1
```

comm (Compare)

The comm command compares two sorted files line by line. It outputs three columns: lines unique to the first file, lines unique to the second file, and lines common to both files.

Syntax

```
comm [options] file1 file2
```

Common Options

- -1: Suppress column 1 (lines unique to file1).
- -2: Suppress column 2 (lines unique to file2).
- -3: Suppress column 3 (lines common to both files).

Examples

1. **Basic Comparison**

```
comm file1.txt file2.txt
```

This command outputs three columns, showing the unique and common lines.

2. **Suppress Column 1**

```
comm -1 file1.txt file2.txt
```

This command suppresses the lines unique to file1.

3. **Suppress Column 2**

```
comm -2 file1.txt file2.txt
```

This command suppresses the lines unique to file2.

4. **Suppress Columns 1 and 2**

```
comm -12 file1.txt file2.txt
```

This command only shows the lines common to both files.

Example Output

For two sorted files file1.txt and file2.txt:

```
comm file1.txt file2.txt
a               # Line unique to file1
b               # Line unique to file1
        c       # Line unique to file2
d       d       # Line common to both files
        e       # Line unique to file2
```

Viewing Files

- **ls**: Lists files and directories.

    - ls -l: Long listing format, shows file permissions, owners, sizes, and modification dates.
    - ls -a: Lists all files, including hidden ones (starting with .).
    - ls -lh: Human-readable sizes, combining -l and -h.
- **cat, more, less**: Viewing file content.
    - cat file.txt: Displays the entire content of file.txt.
    - more file.txt: Pages through file.txt one screen at a time.
    - less file.txt: Similar to more, but allows backward navigation.
- **head, tail**: Viewing the beginning or end of files.
    - head -n 10 file.txt: Shows the first 10 lines of file.txt.
    - tail -n 10 file.txt: Shows the last 10 lines of file.txt.
    - tail -f file.txt: Continuously monitors a file, displaying new lines as they are added.

Disk-Related Commands

- **df (Disk Free)**: Shows disk usage and free space.
    - df -h: Human-readable format, showing sizes in MB, GB, etc.
    - df -T: Shows the type of filesystem.
    - df -i: Shows inode usage instead of block usage.
- **du (Disk Usage)**: Shows the disk usage of files and directories.
    - du -sh: Summarizes disk usage of the current directory in human-readable form.
    - du -h --max-depth=1: Shows the size of each subdirectory within the current directory.
    - du -a: Displays disk usage for all files, not just directories.
- **mount**: Mounts a filesystem.
    - mount: Displays all mounted filesystems.
    - mount /dev/sda1 /mnt: Mounts the device /dev/sda1 to the /mnt directory.
- **umount**: Unmounts a filesystem.
    - umount /mnt: Unmounts the filesystem mounted on /mnt.
- **fdisk**: Used for disk partitioning.
    - fdisk -l: Lists all available partitions and disk devices.
    - fdisk /dev/sda: Opens /dev/sda for partitioning.
- **mkfs (Make Filesystem)**: Formats a disk or partition.
    - mkfs.ext4 /dev/sda1: Formats /dev/sda1 with the ext4 filesystem.
- **fsck (Filesystem Check)**: Checks and repairs filesystems.
    - fsck /dev/sda1: Checks the filesystem on /dev/sda1 for errors.

Checking Disk Free Space

- **df**: As mentioned above, the df command is used to check disk space usage and free space.
    - df -h: Displays disk space usage in a human-readable format.
    - Example output:

bash

```
Filesystem     Size  Used  Avail  Use%  Mounted on
/dev/sda1      50G   20G   28G    42%   /
```

- **du**: Used to check the space used by specific directories and files.
    - du -sh /home: Shows the total size of the /home directory.

Essential Linux Commands understanding Shells

Understanding the essential Linux commands and shells is fundamental for effectively navigating and managing a Linux system. Here's a detailed explanation:

1. What is a Shell?

A **shell** is a command-line interpreter that provides a user interface for the Linux operating system. It allows users to interact with the kernel by typing commands. The shell interprets these commands and passes them to the operating system to execute.

There are several types of shells in Linux, with the most common being:

- **Bash (Bourne Again Shell)**: The default shell for most Linux distributions.
- **Zsh (Z Shell)**: An extended version of Bash with more features.
- **Ksh (Korn Shell)**: A shell that is backward compatible with the Bourne shell, with added features from the C shell.
- **Tcsh**: An enhanced version of the C shell (csh).

2. Essential Linux Commands

Here’s a list of some essential Linux commands, along with explanations of their usage:

File and Directory Management

- **pwd (Print Working Directory)**: Shows the current directory.
    - Example: pwd returns /home/user.
- **ls (List)**: Lists files and directories.
    - ls: Lists all files and directories in the current directory.
    - ls -l: Long format listing (shows permissions, owner, size, and date).
    - ls -a: Lists all files, including hidden files (those starting with .).
- **cd (Change Directory)**: Changes the current directory.
    - cd /home/user: Moves to /home/user directory.
    - cd ..: Moves up one directory level.
    - cd ~: Moves to the home directory.
- **cp (Copy)**: Copies files or directories.

    - cp file.txt /home/user/: Copies file.txt to /home/user/.
    - cp -r dir1 dir2: Recursively copies dir1 to dir2.
- **mv (Move/Rename)**: Moves or renames files or directories.
    - mv file.txt /home/user/: Moves file.txt to /home/user/.
    - mv file.txt newfile.txt: Renames file.txt to newfile.txt.
- **rm (Remove)**: Deletes files or directories.
    - rm file.txt: Deletes file.txt.
    - rm -r dir: Recursively deletes dir and its contents.
- **mkdir (Make Directory)**: Creates a new directory.
    - mkdir newdir: Creates a directory named newdir.
- **rmdir (Remove Directory)**: Deletes an empty directory.
    - rmdir dir: Removes the empty directory dir.

File Viewing and Editing

- **cat (Concatenate)**: Displays the contents of a file.
    - cat file.txt: Shows the contents of file.txt.
- **nano**: A simple text editor within the terminal.
    - nano file.txt: Opens file.txt in the Nano text editor.
- **vi/vim**: Powerful text editors.
    - vim file.txt: Opens file.txt in the Vim text editor.
- **more and less**: Pagers for viewing files.
    - more file.txt: Pages through file.txt one screen at a time.
    - less file.txt: Similar to more, but allows backward navigation.

System Monitoring and Management

- **top**: Displays real-time system information, including CPU and memory usage.
    - top: Opens the task manager to monitor processes.
- **ps (Process Status)**: Displays information about active processes.
    - ps aux: Shows a detailed list of all running processes.
- **kill**: Terminates processes.
    - kill PID: Kills the process with the specified PID (Process ID).
- **df (Disk Free)**: Shows disk usage and available space.
    - df -h: Displays disk space usage in a human-readable format.
- **du (Disk Usage)**: Displays disk usage for files and directories.
    - du -sh: Shows the total size of the current directory.

User Management

- **whoami**: Displays the current logged-in user.

    - whoami: Returns the username of the current session.
- **sudo**: Executes commands with superuser (root) privileges.
    - sudo command: Runs command with elevated privileges.
    - sudo apt update: Updates the package list (on Debian-based systems).
- **chmod (Change Mode)**: Changes file permissions.
    - chmod 755 file.txt: Sets the permissions of file.txt to 755.
- **chown (Change Owner)**: Changes file ownership.
    - chown user:group file.txt: Changes the owner and group of file.txt.

Networking

- **ping**: Tests connectivity to another network device.
    - ping google.com: Sends packets to Google's server to test connectivity.
- **ifconfig**: Configures network interfaces (deprecated, use ip instead).
    - ifconfig: Displays network interfaces and their IP addresses.
- **ip**: A more modern tool for IP address and routing management.
    - ip addr show: Displays all IP addresses assigned to network interfaces.
    - ip route show: Displays the routing table.
- **ssh (Secure Shell)**: Remotely logs in to another computer.
    - ssh user@hostname: Connects to hostname as user.

Searching and Filtering

- **grep**: Searches for patterns in files.
    - grep 'search_term' file.txt: Finds search_term in file.txt.
    - grep -r 'search_term' /path/to/dir: Recursively searches search_term in files under the specified directory.
- **find**: Searches for files and directories.
    - find / -name file.txt: Searches for file.txt starting from the root directory.
- **locate**: Quickly finds files by name (requires an updated database).
    - locate file.txt: Searches for file.txt across the filesystem.
- **awk**: A powerful text processing tool.
    - awk '{print $1}' file.txt: Prints the first column of each line in file.txt.
- **sed (Stream Editor)**: A tool for editing streams of text.
    - sed 's/old/new/g' file.txt: Replaces all instances of old with new in file.txt.

3. Working with Shells

Shells like Bash provide powerful scripting capabilities that go beyond just running commands. Understanding the following concepts is crucial:

- **Shell Scripting**: Writing scripts in shell languages to automate tasks.
    - Example script (script.sh):

bash

```
#!/bin/bash

echo "Hello, World!"
```

Run it with bash script.sh or make it executable with chmod +x script.sh and run it with ./script.sh.

- **Environment Variables**: Used to store data that can be used by the shell and processes.
    - echo $HOME: Displays the home directory of the current user.
    - export VAR=value: Sets a new environment variable VAR with value.
- **Aliases**: Shortcuts for commands.
    - alias ll='ls -l': Creates an alias ll that runs ls -l.
- **Redirection and Pipes**: Used to redirect output and chain commands.
    - >: Redirects output to a file, e.g., ls > file.txt.
    - >>: Appends output to a file, e.g., echo "text" >> file.txt.
    - |: Pipes output from one command to another, e.g., ls | grep "pattern".

Understanding these commands and shell functionalities will give you the foundation to navigate, manage, and automate tasks in a Linux environment.

Processes in Linux

Understanding processes in Linux is crucial for effective system management. Processes in Linux are instances of running programs, and the operating system manages them using various tools and commands. Let's dive into the key concepts related to processes, including process fundamentals, connecting processes with pipes, redirecting input and output, using manual help, and managing background processing.

1. Process Fundamentals

What is a Process?

- A **process** is an instance of a running program. Each process in Linux is identified by a unique Process ID (PID).
- Linux manages processes through a hierarchical structure, where each process has a parent process and may have child processes.

Types of Processes:

- **Foreground Processes**: These are processes that run interactively, requiring user input. The terminal waits until the process completes before accepting new commands.
- **Background Processes**: These processes run independently of the terminal, allowing the terminal to accept new commands while the process runs in the background.
- **Daemon Processes**: These are background processes that start at boot and run continuously, often without user interaction (e.g., web servers, database servers).

Key Commands for Managing Processes:

- **ps (Process Status)**: Displays information about running processes.
    - ps aux: Shows a detailed list of all running processes.

    - ps -ef: Another format to display all processes with additional details.
- **top**: Provides a real-time view of system processes, including CPU and memory usage.
    - top: Opens the task manager-like interface for monitoring processes.
    - Within top, you can press k to kill a process or q to quit.
- **htop**: An improved, interactive version of top (may need to be installed).
    - htop: Launches an enhanced process viewer.
- **kill**: Terminates processes by PID.
    - kill PID: Sends a signal to terminate the process with the specified PID.
    - kill -9 PID: Forcibly kills the process with PID PID (using signal SIGKILL).
- **killall**: Terminates all processes by name.
    - killall processname: Kills all processes with the name processname.
- **pgrep**: Finds the PID of a process by name.
    - pgrep processname: Returns the PID of the process matching processname.
- **nice and renice**: Adjusts the priority of a process.
    - nice -n 10 command: Starts a command with a priority of 10.
    - renice -n 5 PID: Changes the priority of an existing process with PID PID.

2. Connecting Processes with Pipes

Pipes are used to connect the output of one command to the input of another, allowing for complex command chaining.

How Pipes Work:

- **Syntax**: command1 | command2
    - The output of command1 is passed as input to command2.
    - Example: ls -l | grep "pattern": Lists files in long format and filters lines containing "pattern".

Common Uses of Pipes:

- **Filtering Output**:
    - ps aux | grep apache: Filters the list of processes to show only those related to apache.
- **Sorting Data**:
    - ls -l | sort -k 5 -n: Lists files and sorts them by size (5th column) in numerical order.
- **Counting Lines**:
    - cat file.txt | wc -l: Counts the number of lines in file.txt.
- **Chaining Commands**:
    - dmesg | tail -n 20 | less: Shows the last 20 lines of the kernel log and allows paging through the output.

3. Redirecting Input and Output

In Linux, input and output can be redirected to and from files or other commands.

Standard Streams:

- **Standard Input (stdin)**: The input stream, typically from the keyboard (0).
- **Standard Output (stdout)**: The output stream, typically to the terminal (1).
- **Standard Error (stderr)**: The error stream, also typically to the terminal (2).

Redirection Operators:

- **> (Output Redirection)**: Redirects stdout to a file, overwriting the file.
    - command > file.txt: Saves the output of command to file.txt.
- **>> (Append Output Redirection)**: Appends stdout to a file.
    - command >> file.txt: Appends the output of command to file.txt.
- **< (Input Redirection)**: Redirects stdin from a file.
    - command < input.txt: Uses input.txt as the input for command.
- **2> (Error Redirection)**: Redirects stderr to a file.
    - command 2> error.log: Saves error messages to error.log.
- **&> (Combined Redirection)**: Redirects both stdout and stderr to a file.
    - command &> output.log: Saves both standard output and errors to output.log.

Examples of Redirection:

- **Redirecting Output to a File**:
    - ls -l > filelist.txt: Saves the directory listing to filelist.txt.
- **Appending to a File**:
    - echo "New Line" >> filelist.txt: Adds "New Line" to filelist.txt.
- **Redirecting Input from a File**:
    - cat < file.txt: Displays the content of file.txt (similar to cat file.txt).
- **Combining stdout and stderr**:
    - command &> combined.log: Logs both normal output and error messages to combined.log.

4. Manual Help

The man (manual) command is a built-in tool in Linux that provides documentation about commands and utilities.

Using man Pages:

- **Basic Syntax**: man command
    - man ls: Displays the manual page for the ls command.

Navigating man Pages:

- Use Up and Down arrow keys to scroll.
- Press q to quit.
- /search_term: Searches for search_term within the manual page.
- n: Moves to the next occurrence of the search term.

Sections in man Pages:

- man pages are divided into sections:
    - **1**: User commands.
    - **2**: System calls.
    - **3**: Library functions.
    - **5**: File formats and conventions.
    - **8**: System administration commands.

Getting Help for a Specific Section:

- man 5 passwd: Displays the manual page for the passwd file format in section 5.

Other Help Commands:

- **info**: Similar to man, but often provides more detailed documentation.
    - info ls: Displays detailed information about the ls command.
- **--help**: Provides a brief overview of command usage.
    - ls --help: Displays usage information for the ls command.

5. Background Processing

In Linux, you can run processes in the background, allowing you to continue using the terminal while the process runs.

Running a Process in the Background:

- **Appending & to a command**:
    - command &: Runs command in the background.
    - Example: sleep 60 &: Runs the sleep command in the background, pausing for 60 seconds.

Managing Background Jobs:

- **jobs**: Lists all background jobs.
    - jobs: Displays current jobs, showing their job ID and status.
- **fg (Foreground)**: Brings a background job to the foreground.
    - fg %1: Brings job number 1 to the foreground.
- **bg (Background)**: Resumes a stopped job in the background.
    - bg %1: Resumes job number 1 in the background.
- **Stopping a Job**:
    - Press Ctrl + Z: Suspends the currently running foreground process.
- **Killing a Job**:
    - kill %1: Kills job number 1.

Example of Background Processing:

- **Run a Command in the Background**:

    - find / -name '*.log' > logs.txt &: Searches for all .log files and saves the output to logs.txt while allowing the terminal to be used for other commands.

- **View and Manage Jobs**:
    - Run jobs to see the list of background jobs.
    - Use fg %1 to bring job 1 to the foreground if you need to interact with it.

Understanding these concepts and commands gives you control over processes in a Linux environment, allowing you to efficiently manage tasks, automate workflows, and troubleshoot issues as they arise.