Index

Introduction to Linux

- Glimpse of Linux 1
- History of Linux 1
- Types of Linux 2

Basic Commands & Usage

- Basic Linux commands 7
- Keyboard shortcuts 8

Files & Directories

- Filesystem 12
- Permissions 13
- Wildcard 14
- Locate command 16
- Find command 18

Text Editors & String Manipulation

- Vim vs Nano 22
- String Manipulation Tools 25

Command Redirection & Execution

- Linux Redirectors 33
- xargs Command 35
- tar command 37

System Information & Management

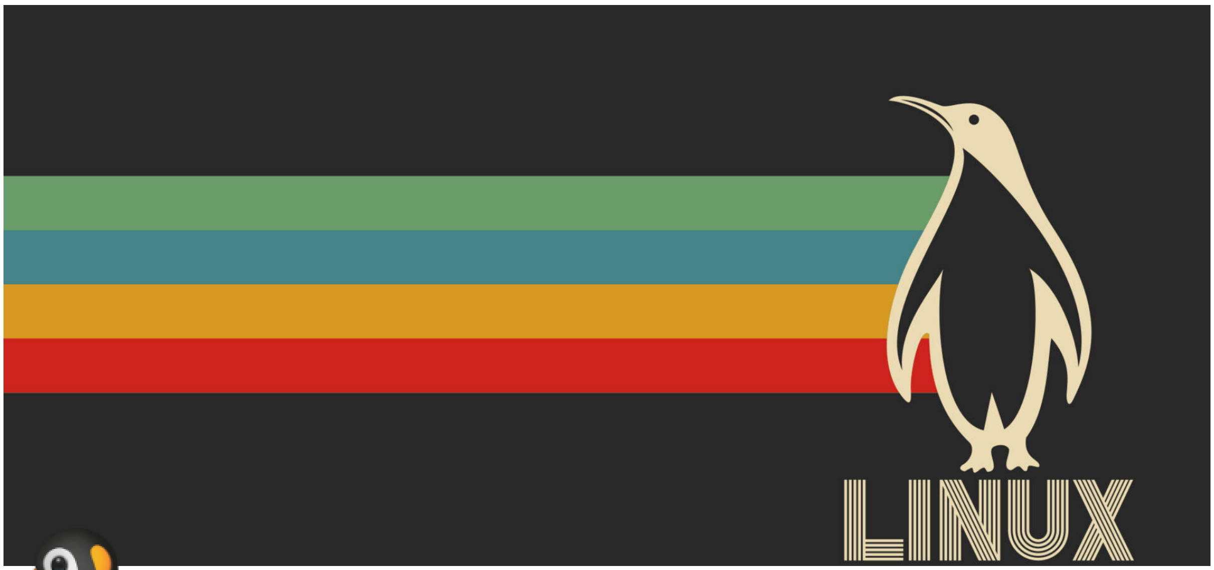
- System Detail Commands 40
- Devices in Linux 44
- Boot Process in Linux 48

Processes, Services & Package Management

- Cron jobs 59
- Curl 64

Networking & Security

- Networking command 66
- SSH 73
- Network Sharing 75
- Http Server 83



🐧

Linux

1. Introduction

Glimpse of Linux 🐧

What is Linux?

Linux is a free and open-source operating system based on the UNIX operating system. The core component of Linux is the **Linux kernel**, which was created by Linus Torvalds in 1991. The kernel is responsible for managing the system's hardware and providing essential services for all other parts of the operating system.

Key Features of Linux

- **Open Source**: The source code of Linux is freely available for anyone to view, modify, and distribute.
- **Security**: Known for its strong security model and robust permissions system.
- **Stability and Performance**: Linux is highly stable and efficient, making it a popular choice for servers and critical applications.

- **Flexibility**: Can be run on a wide range of hardware from personal computers to mainframes and embedded systems.
- **Community Support**: Large and active community providing support, updates, and a wide variety of software applications.

History of Linux ⌛

1. Origins of UNIX and Early Influences (1960s-1980s)

- **UNIX**:
    - Developed in the late 1960s and early 1970s at AT&T's Bell Labs by Ken Thompson, Dennis Ritchie, and others.
    - Influential in the development of operating systems due to its portability, multi-tasking, and multi-user capabilities.
    - Source code was freely shared with academic institutions, which led to widespread adoption and variations.
- **Minix**:
    - Created by Andrew S. Tanenbaum in 1987 as a teaching tool.
    - A small, UNIX-like operating system that ran on IBM PC-compatible computers.
    - Influenced Linus Torvalds in the development of Linux.

2. Creation of Linux (1991)

- **Linus Torvalds**:
    - A Finnish computer science student at the University of Helsinki.
    - Inspired by Minix and the need for a free and open operating system kernel for personal computers.
- **Initial Announcement**:
    - On August 25, 1991, Torvalds announced his project on the comp.os.minix newsgroup, seeking feedback from other developers.
    - Released the first version (0.01) on September 17, 1991.

3. Development and Growth (1990s)

- **Early Contributions**:
    - Early versions of Linux were distributed under the GNU General Public License (GPL), allowing anyone to use, modify, and distribute the software.
    - Attracted a global community of developers who contributed code, features, and improvements.
- **Key Milestones**:
    - **1992**: Linux 0.12, the first version under the GPL.
    - **1993**: The first Linux distributions, such as Slackware and Debian, emerged.
    - **1994**: Linux 1.0 released with the first production-ready kernel.
    - **1996**: Linux 2.0 released, introducing SMP (symmetric multi-processing) support and making Linux suitable for enterprise environments.

4. Commercial Adoption and Enterprise Use (2000s)

- **Red Hat and SUSE**:
    - Early commercial distributions that provided support, training, and certification.
    - Played a significant role in enterprise adoption of Linux.
- **Major Milestones**:
    - **2003**: Red Hat Enterprise Linux (RHEL) launched, targeting enterprise customers.
    - **2004**: Ubuntu released by Canonical, aiming to make Linux more user-friendly and accessible to a broader audience.

5. Technological Advances and Modern Era (2010s-Present)

- **Linux in Cloud and Mobile**:
    - Linux became the foundation for many cloud computing platforms and services, such as AWS, Google Cloud, and Microsoft Azure.

    - **Android**, developed by Google and based on the Linux kernel, became the dominant mobile operating system.
- **Containerization and DevOps**:
    - Technologies like Docker and Kubernetes, which are based on Linux, revolutionized software development and deployment.
    - Enabled the rise of DevOps practices and microservices architecture.
- **Continued Development**:
    - The Linux kernel continues to evolve, with contributions from a diverse community of individual developers, corporations, and organizations.
    - Linus Torvalds remains the chief maintainer and oversees the development process.

6. Community and Open Source Culture

- **Collaborative Development**:
    - The Linux community embodies the principles of open-source software: transparency, collaboration, and meritocracy.
    - Contributions come from individuals, educational institutions, and major tech companies like IBM, Intel, and Google.
- **Support and Documentation**:
    - Extensive online documentation, forums, and user groups help users and developers troubleshoot and improve their Linux experience.

7. Impact and Legacy

- **Server and Supercomputing**:
    - Linux is the preferred operating system for servers, supercomputers, and high-performance computing clusters.
    - Known for its stability, security, and scalability.
- **Embedded Systems**:
    - Widely used in embedded systems, including routers, smart TVs, and IoT devices.
- **Education and Innovation**:

- Continues to be a valuable tool in computer science education and research.
- Drives innovation in software development, networking, and cybersecurity.

Types of Linux

Linux distributions (distros) are versions of Linux that include the Linux kernel, system software, and applications. Different distributions are tailored to various needs, from general-purpose desktop use to specialized server environments.

1. General-Purpose Distributions

Distribution	Description
Ubuntu	User-friendly, popular for desktops and servers. Derived from Debian.
Fedora	Sponsored by Red Hat, known for cutting-edge features and technologies.
Debian	Known for its stability and extensive software repository.
Linux Mint	Based on Ubuntu/Debian, designed for ease of use and to provide a full multimedia experience out of the box.
openSUSE	Sponsored by SUSE, known for its YaST configuration tool and strong focus on user-friendliness and security.

2. Enterprise Distributions

Distribution	Description
Red Hat Enterprise Linux (RHEL)	Commercial distribution, widely used in enterprise environments. Known for stability and long-term support.
SUSE Linux Enterprise Server (SLES)	Enterprise-grade distribution, known for its strong support for virtualization and cloud environments.
Oracle Linux	Derived from RHEL, optimized for Oracle software and databases.

3. Lightweight Distributions

Distribution	Description

Puppy Linux	Extremely lightweight, designed to run on older hardware or low-resource environments.
Lubuntu	Based on Ubuntu, uses the LXQt desktop environment for low resource usage.
Tiny Core Linux	Very small footprint, highly modular, and customizable.

4. Security-Focused Distributions

Distribution	**Description**
Kali Linux	Designed for penetration testing and security auditing. Comes with numerous pre-installed security tools.
Parrot Security OS	Similar to Kali, focused on security, privacy, and development tools.
Qubes OS	Focuses on security through isolation, using virtualization to compartmentalize different tasks.

5. Server Distributions

Distribution	**Description**
CentOS	Community-supported distribution derived from RHEL, suitable for servers.
Ubuntu Server	Server edition of Ubuntu, known for ease of use and deployment.
Arch Linux	Lightweight and flexible, users can build a custom server environment from the ground up.

6. Specialized Distributions

Distribution	**Description**
Raspberry Pi OS (formerly Raspbian)	Optimized for the Raspberry Pi hardware, based on Debian.
Tails	Security-focused, designed to preserve privacy and anonymity. Runs from a USB stick or DVD.
Android-x86	Port of the Android operating system for x86 processors, enabling Android to run on standard PCs.

2. Basic Commands & Usage

Basic Linux commands

These are some of the most common and used Linux commands. you can refer this as commands Cheatsheet.

Navigation

Command	Description
`cd`	Change directory
`cd /path/to/directory`	Change to specified directory
`cd .`	Current directory
`cd ..`	Parent directory
`cd ~`	Home directory
`cd -`	Previous directory
`ls`	List files and directories
`ls -l`	Long format
`ls -a`	Including hidden files
`ls -i`	Display inode numbers
`ls -lh`	Human-readable file sizes (e.g., KB, MB, GB)
`ls -r`	Reverse the order of the sort to list files in reverse
`ls -R`	Recursively list subdirectories
`ls -lS`	Sort files by size, largest first
`ls -d */`	List only directories
`pwd`	Print working directory

File Management

Command	Description
`cp`	Copy files or directories
`cp file1 file2`	Copy file1 to file2

`cp -r directory1 directory2`	Copy directory1 to directory2 recursively
`cp -i`	Protect from overwriting a file
`mv`	Move or rename files/directories
`mv file1 newfile`	Rename file1 to newfile
`mv file1 /path/to/directory/`	Move file1 to specified directory
`rm`	Remove files or directories
`rm file1`	Remove file1
`rm -r directory1`	Remove directory1 recursively
`rm -i directory1`	Prompt before removing files or directories
`touch`	Create an empty file
`touch filename`	Create an empty file named filename

File Viewing and Editing

Command	Description
`cat`	Display file content
`cat filename`	Display the content of filename
`more`	Display content one screen at a time
`more filename`	Display the content of filename one screen at a time
`less`	Display content one screen at a time
`less filename`	Display the content of filename one screen at a time
`nano`	Text editor
`nano filename`	Edit filename using nano
`vim`	Text editor
`vim filename`	Edit filename using vim

System Information

Command	Description
`uname`	Print system information
`uname -a`	Print all system information
`hostname`	Print or set system name
`top`	Display system processes
`htop`	Display system processes (interactive)

User Management

Command	Description
`whoami`	Print the current username
`who`	Display information about users currently logged in
`useradd`	Add a new user
`passwd`	Change user password
`usermod`	Modify user account

Package Management

Command	Description
`apt` or `apt-get`	Package management on Debian-based systems
`sudo apt update`	Update package lists
`sudo apt install package-name`	Install a package named package-name
`sudo apt remove package-name`	Remove a package named package-name
`yum` or `dnf`	Package management on Red Hat-based systems

Process Management

Command	Description
`ps`	Display information about running processes
`ps aux`	Detailed list of running processes
`kill`	Terminate a process
`kill PID`	Terminate the process with the specified PID

Networking

Command	Description
`ifconfig`	Display network configuration
`ip`	Display network configuration
`ping`	Test network connectivity
`ping example.com`	Test connectivity to example.com
`netstat`	Display network connections and routing tables
`traceroute`	Trace the route to a network host

File Permissions

Command	Description
`chmod`	Change file permissions
`chmod +x filename`	Make file executable
`chown`	Change file owner
`chown user:group filename`	Change owner of filename to user and group

System Logs

Command	Description
`dmesg`	Display kernel messages
`journalctl`	Query and display messages from the journal

Keyboard shortcuts ⌨️

Shortcut	Action
`Ctrl + A`	Move to the beginning of the line.
`Ctrl + E`	Move to the end of the line.
`Ctrl + U`	Cut/delete the line before the cursor.
`Ctrl + K`	Cut/delete the line after the cursor.
`Ctrl + W`	Cut/delete the word before the cursor.
`Ctrl + Y`	Paste the last cut text.
`Ctrl + L`	Clear the terminal screen.
`Ctrl + C`	Cancel the current command.
`Ctrl + Z`	Suspend the current command.
`Ctrl + R`	Search the command history.
`!!`	Repeat the last command.
`!n`	Execute the `n` th command in history.

General Shortcuts

Shortcut	Description
`Alt + F2`	Open the run command dialog
`Alt + Tab`	Switch between open applications
`Ctrl + Alt + T`	Open a new terminal window
`Ctrl + Alt + L`	Lock the screen
`Alt + F4`	Close the current window

File Management Shortcuts

Shortcut	Description
`Ctrl + N`	Open a new file manager window
`Ctrl + T`	Open a new tab in the file manager
`Ctrl + Shift + N`	Create a new folder
`Ctrl + C`	Copy selected files
`Ctrl + V`	Paste copied files
`Ctrl + X`	Cut selected files
`Ctrl + A`	Select all files
`F2`	Rename the selected file

Text Editor Shortcuts (e.g., Nano)

Shortcut	Description
`Ctrl + O`	Write out (save) the file
`Ctrl + X`	Exit the editor
`Ctrl + K`	Cut the selected text
`Ctrl + U`	Paste the cut text
`Ctrl + W`	Search within the file

3. Files & Directories

Filesystem 📂

The Linux filesystem follows the Filesystem Hierarchy Standard (FHS), which defines the directory structure and directory contents in Unix-like operating systems. Understanding the Linux filesystem is crucial for system administration, development, and troubleshooting.

> key points
>
> - **Everything is a File**: In Linux, almost everything (including hardware devices and processes) is treated as a file.
> - **Hierarchical Structure**: The Linux filesystem is organized in a tree-like hierarchy, with the root directory (/) at the top.
> - **Mount Points**: Devices and partitions are integrated into the filesystem tree through mount points.

Root Directory (/)

The root directory is the top-level directory in the Linux filesystem. All other directories and files are located under the root directory.

Standard Directories

Directory	Description
/bin	Essential command binaries needed for booting and single-user mode.
/boot	Static files of the bootloader, such as the kernel and initrd images.
/dev	Device files representing hardware components (e.g., /dev/sda for a disk).
/etc	Configuration files and system-wide configuration scripts.

/home	User home directories. Each user has a subdirectory here (e.g., /home/user).
/lib	Essential shared libraries and kernel modules.
/media	Mount points for removable media like CDs, DVDs, and USB drives.
/mnt	Temporary mount points for filesystems.
/opt	Optional application software packages.
/proc	Virtual filesystem providing process and kernel information.
/root	Home directory for the root user.
/run	Runtime data for processes started since the last boot.
/sbin	Essential system binaries (e.g., system administration commands).
/srv	Data for services provided by the system (e.g., web server data).
/sys	Virtual filesystem providing device and kernel information.
/tmp	Temporary files. Cleared on reboot.
/usr	Secondary hierarchy for read-only user data; contains applications and libraries.
/var	Variable data files (e.g., logs, databases, email spools).

Filesystem Types

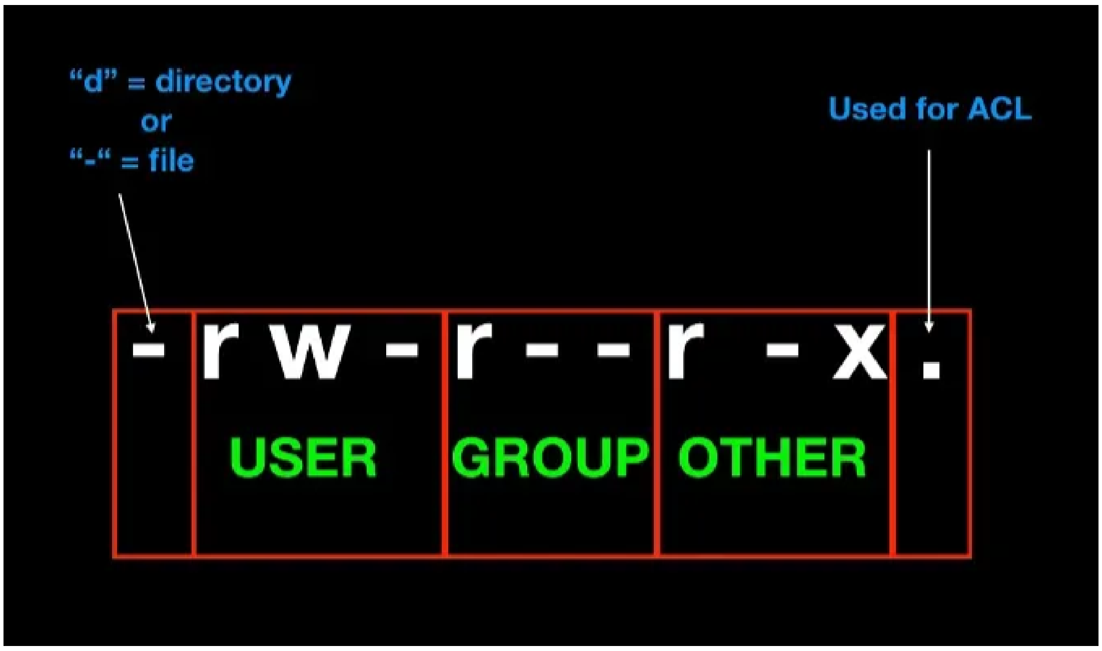
- **ext4**: Default on most Linux distributions; journaling, fast, and reliable.
- **XFS**: High-performance, scalable, great for large files and databases.
- **Btrfs**: Modern, supports snapshots, RAID, and advanced features for data integrity.
- **ZFS**: Powerful, scalable, with RAID and built-in compression; often used in enterprise setups.
- **FAT32**: Simple, widely compatible but limited to 4GB files, commonly used on USB drives.
- **exFAT**: Supports larger files than FAT32, often used on external drives.
- **NTFS**: Used by Windows, read/write supported on Linux but not native.

Filesystem Commands

Command	**Description**	**Example**
ls	List directory contents	`ls -l` (detailed listing)

cd	Change the current directory	`cd /home/user`
pwd	Print the current working directory	`pwd`
mkdir	Create a new directory	`mkdir new_folder`
rmdir	Remove an empty directory	`rmdir empty_folder`
rm	Remove files or directories	`rm file.txt` (file)
cp	Copy files or directories	`cp file1.txt file2.txt`
mv	Move or rename files or directories	`mv oldname.txt newname.txt`
ln	Create hard and symbolic links	`ln -s target symlink`
df	Report filesystem disk space usage	`df -h` (human-readable format)
du	Estimate file space usage	`du -sh directory`
mount	Mount a filesystem	`mount /dev/sda1 /mnt`
umount	Unmount a filesystem	`umount /mnt`
fsck	Check and repair a filesystem	`fsck /dev/sda1`
mkfs	Create a new filesystem	`mkfs.ext4 /dev/sda1`
tune2fs	Adjust tunable filesystem parameters on ext2/3/4	`tune2fs -c 20 /dev/sda1` (set check count)

Permissions 🔐



To view and modify file permissions, you can use the `ls -l` command to list files and their permissions and the `chmod` command to change permissions.

- `chmod u+x file.txt`: Adds execute permission for the owner of the file.
- `chmod go-w file.txt`: Removes write permission for the group and others.
- `chmod a+r file.txt`: Adds read permission for all users (owner, group, and others).
- There are numerical value for permissions :
  - 4: read permission
  - 2: write permission
  - 1: execute permission
  - `$ chmod 755 myfile` : 755 covers the permissions for all sets. The first number (7) represents user permissions, (5) represents group permissions and the last (5) represents other permissions.

Wildcard

1. `*` (Asterisk): Matches any sequence of characters (including none).

- Example: `.txt` matches all files ending with ".txt".

2. `?` (Question Mark): Matches any single character.
   - Example: `file?.txt` matches "file1.txt", "fileA.txt", etc.

3. `[ ]` (Square Brackets): Matches any single character within the specified range or set.

- Example: `[aeiou]` matches any single vowel.

4. `[^ ]` (Caret Inside Square Brackets): Matches any single character not in the specified range or set.
   - Example: `[^0-9]` matches any non-digit character.
5. `{ }` (Curly Braces): Specifies a list of alternatives.
   - Example: `{jpg,png}` matches either "file.jpg" or "file.png".
6. `!` (Exclamation Mark): Represents negation or exclusion.
   - Example: `!*.log` matches all files except those ending with ".log".
7. `()` (Parentheses): Groups patterns together.
   - Example: `(file|document)*.txt` matches "file1.txt", "documentA.txt", etc.

locate command 📌

The `locate` command in Linux is used to quickly find files and directories by name. It relies on a database that is periodically updated, making it faster than searching through the file system in real-time. Here's a detailed overview:

Overview of `locate` Command

- **Purpose**: Quickly find files and directories by querying a pre-built database of file names.
- **Usage**: Useful for locating files without having to perform a full file system search.

Basic Syntax

```
locate [options] [pattern]
```

- `pattern`: The search term or pattern to look for in the file names.

Common Options

- `i`: Perform a case-insensitive search.

  ```
  locate -i pattern
  ```

- `r`: Use a regular expression to search for patterns.

  ```
  locate -r 'pattern.*'
  ```

- `n`: Limit the number of results returned.

  ```
  locate -n 5 pattern
  ```

- `c`: Count the number of matching entries instead of displaying them.

  ```
  locate -c pattern
  ```

- `-regex`: Same as `r`, used to search with regular expressions.

  ```
  locate --regex 'pattern.*'
  ```

- `-database`: Use a specific database file instead of the default.

  ```
  locate --database /path/to/database pattern
  ```

How `locate` Works

1. **Database Update**:
   - The `locate` command relies on a database (`mlocate.db`) that is updated regularly by a system cron job, typically using the `updatedb` command.
   - The database contains a list of files and directories on the system.
2. **Searching**:
   - When you run `locate`, it searches through the database rather than the file system, which is much faster.

3. **Updating the Database**:
    - To manually update the database, use the `updatedb` command:

      ```
      sudo updatedb
      ```

find command 🔎

The `find` command in Linux is a powerful tool used to search for files and directories in a directory hierarchy based on various criteria. Unlike `locate`, which searches a pre-built database, `find` performs real-time searches directly on the filesystem.

Overview of `find` Command

- **Purpose**: To search for files and directories based on name, type, size, permissions, modification time, and other attributes.
- **Usage**: Useful for locating files and directories with specific attributes or for performing batch operations on files.

Basic Syntax

```
find [path] [options] [expression]
```

- `path`: The directory path to start the search. Use `.` for the current directory.
- `options`: Flags to modify the search behavior.
- `expression`: Criteria to match files and directories.

Common Options and Expressions

- `name`: Search for files or directories by name.

  ```
  find /path/to/search -name "filename"
  ```

- `iname`: Case-insensitive search for files or directories by name.

```
find /path/to/search -iname "filename"
```

- `type`: Search by type (e.g., file, directory, symbolic link).
    - `f` for regular files
    - `d` for directories
    - `l` for symbolic links

```
find /path/to/search -type f -name "filename"
```

- `size`: Search by file size.
    - Example: `+1M` for files larger than 1 MB, `100k` for files smaller than 100 KB

```
find /path/to/search -size +1M
```

- `mtime`: Search by modification time.
    - `+n` for more than `n` days ago
    - `n` for less than `n` days ago
    - `n` for exactly `n` days ago

```
find /path/to/search -mtime -7
```

- `perm`: Search by file permissions.
    - Example: `644` for files with permissions `rw-r--r--`

```
find /path/to/search -perm 644
```

- `user`: Search for files owned by a specific user.

```
find /path/to/search -user username
```

- `group`: Search for files owned by a specific group.

```
find /path/to/search -group groupname
```

- `exec`: Execute a command on each file found.
    - `{}` is replaced by the current file
    - `\\;` terminates the command

```
find /path/to/search -name "*.txt" -exec ls -l {} \\;
```

- `print`: Print the file names (default action if no other action is specified).

```
find /path/to/search -name "*.txt" -print
```

- `delete`: Delete files that match the criteria.

```
find /path/to/search -name "*.tmp" -delete
```

- `empty`: Search for empty files or directories.

```
find /path/to/search -empty
```

- `maxdepth`: Limit the search to a specific number of directory levels.

```
find /path/to/search -maxdepth 2 -name "filename"
```

- `mindepth`: Start the search from a specific number of directory levels.

```
find /path/to/search -mindepth 2 -name "filename"
```

Examples

- **Find all `.txt` files in the current directory and its subdirectories**:

```
find . -name "*.txt"
```

- **Find all directories named `backup`**:

```
find / -type d -name "backup"
```

- **Find all files larger than 100 MB**:

  ```
  find /path/to/search -size +100M
  ```

- **Find files modified in the last 7 days**:

  ```
  find /path/to/search -mtime -7
  ```

- **Find files owned by the user `john` and execute `ls -l` on each**:

  ```
  find /path/to/search -user john -exec ls -l {} \\;
  ```

- **Delete all `.tmp` files**:

  ```
  find /path/to/search -name "*.tmp" -delete
  ```

4. Text Editors & String Manipulation

vim VS nano 📄

Vim and Nano: A Detailed Overview

When it comes to text editors in the Linux/Unix world, **Vim** and **Nano** are two of the most commonly used. Each has its own strengths and is suited for different types of users and use cases. Below is a detailed overview of both editors.

Vim

Overview:

- **Vim** stands for **Vi Improved** and is an enhanced version of the older **Vi** text editor.
- It is a highly configurable and powerful text editor used by developers, system administrators, and power users.
- Vim is known for its steep learning curve but offers efficiency and speed once mastered.

Key Features:

- **Modes:** Vim operates in several modes:
  - **Normal Mode:** For navigating and manipulating text.
  - **Insert Mode:** For inserting text.
  - **Visual Mode:** For selecting blocks of text.
  - **Command Mode:** For executing commands.
- **Customizability:** Extensive customization via `.vimrc` configuration files.
- **Plugins:** A vibrant ecosystem of plugins that extend functionality (e.g., syntax highlighting, code autocompletion).
- **Keyboard Shortcuts:** Efficient editing through powerful keyboard commands, reducing the need for mouse interaction.
- **Search and Replace:** Advanced search capabilities, including regular expression support.

Basic Commands:

- **Opening a file:** `vim filename`
- **Insert mode:** Press `i` to enter insert mode.
- **Save and exit:** Press `Esc`, then type `:wq` and press `Enter`.
- **Exit without saving:** Press `Esc`, then type `:q!` and press `Enter`.
- **Undo:** Press `u` in normal mode.
- **Redo:** Press `Ctrl + r` in normal mode.
- **Copy (yank) text:** Press `y` followed by a movement command.
- **Paste text:** Press `p`.
- **Search:** Press `/` followed by the search term.

Pros:

- Extremely powerful and flexible once mastered.
- Efficient for large-scale editing tasks and repetitive text manipulation.
- Extensive plugin support makes Vim suitable for a wide range of tasks, including coding.

Cons:

- Steep learning curve for beginners.
- Not as user-friendly or intuitive for new users.

Vim Cheat Sheet

Explore vim.rtorr.com for an extensive Vim cheat sheet, offering clear, concise commands and shortcuts for Vim users. Whether you're a beginner or an experienced developer, find tips and tricks to enhance your coding efficiency in Vim.

https://vim.rtorr.com/

Nano

Overview:

- **Nano** is a simple and easy-to-use text editor that is more accessible for beginners compared to Vim.

- It is designed to be straightforward, with a focus on simplicity and ease of use.

Key Features:

- **Ease of Use:** Nano is beginner-friendly, with commands listed at the bottom of the editor.
- **No Modes:** Unlike Vim, Nano operates in a single mode, making it simpler for new users.
- **Keyboard Shortcuts:** Intuitive keyboard shortcuts for basic operations (e.g., save, exit, cut, paste).
- **Search and Replace:** Simple search and replace functionality.
- **Configurable:** Customizable via configuration files, though not as extensively as Vim.

Basic Commands:

- **Opening a file:** `nano filename`
- **Save:** Press `Ctrl + O`, then press `Enter`.
- **Exit:** Press `Ctrl + X`.
- **Cut (delete) a line:** Press `Ctrl + K`.
- **Paste a line:** Press `Ctrl + U`.
- **Search:** Press `Ctrl + W` and then enter the search term.

Pros:

- Very user-friendly and easy to learn, even for users with no prior experience.
- Simple interface with on-screen command reminders.
- No need to memorize complex commands or switch between modes.

Cons:

- Less powerful than Vim; limited features and customization options.
- Not as efficient for large-scale or complex text editing tasks.

String Manipulation

1. `strings` : used to extract readable text strings from binary files. When you run `strings` on a file, it scans through the binary data looking for sequences of printable characters and displays them. This is often useful for extracting human-readable information from compiled executables or other binary files.

   ```
   strings data.txt
   ```

2. `env` : Environment variables in Linux are dynamic values that can affect the behavior of processes and programs running in the operating system.

   ```
   # display env variables

   $ env

   $ echo $PATH

   /usr/local/sbin:/usr/local/bin:/usr/sbin:/bin

   # This returns a list of paths separated by a colon that
   your system searches when it runs a command.

   # Modify the path variable

   export PATH=$PATH:/path/to/new/directory
   ```

3. `cut` : It is used for cutting out sections from each line of a file or from piped data.

   - `c, --characters=LIST` : Select only these characters.
   - `f, --fields=LIST` : Select only these fields.
   - `d, --delimiter=DELIM` : Use DELIM instead of TAB for field delimiter.

```
# Extract 5th character from each line
cut -c 5 sample.txt

# Extract characters 3-7 and 12-16 from each line of a fil
cut -c 3-7,12-16 filename.txt

# Extract the first and third fields (using a comma as the
# By default it 'd' is TAB
cut -f 1,3 -d , filename.csv

# Extract the second field (using a comma as the delimiter
# the output of another command:
echo "John,Doe,25" | cut -f 2 -d ,
```

4. `head` : By default the head command will show you the first 10 lines in a file.
   - The -n flag stands for number of lines.

```
head -n 15 /var/log/syslog # diaplay first 15 lines
head -c 15 /var/log/syslog # display first 15 characters
```

5. `tail` : It is similar to head command as it prints the last 10 lines of a file

```
tail -3 filename # display last 3 lines
tail +3 filename # skips first 2 lines
```

6. `paste` : The paste command is similar to the cat command, it merges lines together in a file.
   - **EXAMPLES**

```
# Display the first 5 lines (head) and the last 5 lines (t
paste <(head -n 5 filename.txt) <(tail -n 5 filename.txt)

# Display the first 3 lines (head) and the last 3 lines (t
paste <(ls | head -n 3) <(ls | tail -n 3)
```

7. `expand` : To converts your TABs to spaces, use the expand command.(it doesn’t remove space).

8. `unexpand` : Opposite of expand

```
expand sample.txt > result.txt
unexpand -a result.txt

tr -s '\t' ' ' < input.txt > output.txt #It will reove tab
tr -s '\n' ' ' < sample.txt # This change the newline into
tr -s ';' ' ' < sample.txt # This will change semicolon to

NOTE- This will not change them temporarily, you need to s
```

9. `sort` : sort a file
   - `-r, --reverse` : Reverse the result of comparisons.
   - `-n, --numeric-sort` : Sort numerically.
   - `-u, --unique` : Output only unique lines.

```
sort filename.txt

# sort the file and '-m' merge the file
sort -m file1.txt file2.txt file3.txt > sorted_output.txt
```

10. `tr` : The tr (translate) command allows you to translate a set of characters into another set of characters.

```
tr a-z A-Z # coverts the lowecase to uppercase

hello

HELLO

echo "123abc456" | tr -d '0-9' # Delete digits from stri
ng
```

11. `split` : This will split it into different files, by default it will split them once they reach a 1000 line limit.

```
# his splits input.txt into files each containing 100 line
split -l 100 input.txt

-l NUMBER: Split the file into chunks of NUMBER lines.
-b SIZE[K|M|G]: Split the file into chunks of specified si
```

12. `join` : The join command allows you to join multiple files together by a common field:

```
$ join -1 2 -2 1 file1.txt file2.txt

1 John Doe

2 Jane Doe

3 Mary Sue
```

13. `uniq` : The `uniq` (unique) command is another useful tool for parsing text.

```
uniq reading.txt # remove duplicates

uniq -c reading.txt # get the count of how many occurrence

uniq -u reading.txt # get unique values (occur only one ti

uniq -d reading.txt # get duplicate values

Note : uniq does not detect duplicate lines unless they ar

# To overcome this limitation of uniq we can use sort in c
sort reading.txt | uniq
```

14. `wc` : The `wc` command stands for "word count" and is used to count the number of lines, words, and characters in a file.

```
wc myfile.txt

-l: Count lines.
```

```
-w: Count words.
-c: Count characters.
```

15. `nl` : used to number lines in a file

```
$ nl file1.txt

1. i
2. like
3. turtles
```

Grep

To find a certain pattern in a file or text, you can use the `grep` command

```
grep "pattern" filename.txt

-i: Perform a case-insensitive search.
-n: Display line numbers along with matched lines.
-r or -R: Recursively search subdirectories.
-w: Match whole words.
-c: Display only the count of matching lines.
-v: Invert the match, displaying lines that do not match the

# 'grep' with other cmd.
find /home/user -type f -name "*.txt" | grep "important" # fi

grep -iRl [directory path/keyword] # Find files with a specif
```

`-include="*.password"` : Specifies the file pattern to include in the search.

For example, to find all files with the `.password` extension that contain the pattern "secret" in the home directory, you would use:

```
grep -rl --include="*.password" "secret" ~
```

1. **egrep**:
   - `egrep` stands for "extended grep".
   - It is essentially the same as running `grep -E`.
   - `egrep` uses extended regular expressions (ERE) for pattern matching, which allows for more powerful and flexible pattern matching compared to basic regular expressions.
   - Extended regular expressions include additional features such as the use of metacharacters like `+`, `?`, `|`, and parentheses `()` without escaping.
   - Example: Searching for lines containing either "apple" or "banana" in a file:

     ```
     egrep 'apple|banana' file.txt
     ```

2. **fgrep**:
   - `fgrep` stands for "fixed grep".
   - It is essentially the same as running `grep -F`.
   - `fgrep` interprets the pattern as a set of fixed strings, rather than regular expressions.
   - This means that `fgrep` searches for literal occurrences of the specified strings in the input text, without interpreting any metacharacters or regular expressions.
   - It is faster than `grep` and `egrep` when searching for fixed strings, as it does not need to parse and interpret regular expressions.
   - Example: Searching for lines containing the exact string "apple" in a file:

     ```
     fgrep 'apple' file.txt
     ```

AWK

It is a powerful and versatile text processing tool in Linux. It is primarily used for pattern scanning and processing. AWK is abbreviated after it's creators (Aho, Weinberger, and Kernighan).

Note: The $0 variable points to the whole line. *Also, make sure to use single quotes('') to specify patterns, awk treats double quotes("") as a raw string. To use double quotes make sure that you escape the ($) sign(s) with a backslash (\) each, to make it work properly.*

Examples:

1. **Print Specific Columns:**

   ```
   awk '{print $1, $3}' filename
   ```

   This prints the first and third columns of each line in the file.

2. **Print Lines Matching a Pattern:**

   ```
   awk '/pattern/ {print}' filename
   ```

   This prints all lines in the file that contain the specified pattern.

3. **Calculate and Print Column Sum:**

   ```
   awk '{sum += $1} END {print "Sum =", sum}' filename
   ```

   This calculates and prints the sum of the values in the first column.

4. **Print Lines Longer Than a Certain Length:**

   ```
   awk 'length($0) > 10' filename
   ```

   This prints lines in the file that are longer than 10 characters.

5. **Print Fields with a Delimiter:**

   ```
   awk -F':' '{print $1, $3}' /etc/passwd
   ```

This prints the first and third fields of the `/etc/passwd` file using `:` as the field separator.

6. **Using Conditional Statements:**

   ```
   awk '{ if ($1 > 50) print $1, "is greater than 50"; else
   print $1, "is not greater than 50" }' filename
   ```

   This uses an `if-else` statement to check if the value in the first column is greater than 50.

7. **Print Unique Lines:**

   ```
   awk '!seen[$0]++' filename
   ```

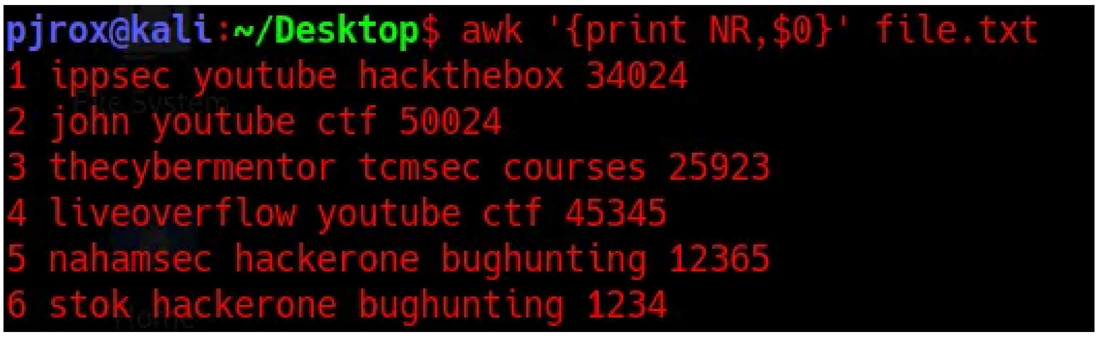
   This prints only unique lines in the file, removing duplicates.

8. **Print the Number of Lines in a File:**

   ```
   awk 'END {print NR}' filename
   ```

   This prints the total number of lines in the file.

   - **NR:** (Number Record) is the variable that keeps count of the rows after each line's execution... You can use NR command to number the lines (`awk '{print NR,$0}' file.txt`). Note that awk considers rows as records.

     ```
     pjrox@kali:~/Desktop$ awk '{print NR,$0}' file.txt
     1 ippsec youtube hackthebox 34024
     2 john youtube ctf 50024
     3 thecybermentor tcmsec courses 25923
     4 liveoverflow youtube ctf 45345
     5 nahamsec hackerone bughunting 12365
     6 stok hackerone bughunting 1234
     ```

9. **Print Only Lines between Patterns:**

   ```
   awk '/start_pattern/, /end_pattern/' filename
   ```

   This prints lines in the file between the specified start and end patterns.

5. Command Redirection & Execution

Linux Redirectors ➡️

1. `>` **(Output Redirection):**
   - Redirects the standard output (**stdout**) of a command to a file. If the file exists, it is overwritten.

   ```
   command > output.txt
   ```

2. `>>` **(Append Output):**
   - Appends the standard output of a command to a file. If the file doesn't exist, it is created.

   ```
   command >> output.txt
   ```

3. `<` **(Input Redirection):**
   - Redirects the standard input (**stdin**) of a command from a file.

   ```
   command < input.txt
   ```

4. `|` **(Pipe):**
   - Redirects the output of one command as the input to another command.

   ```
   command1 | command2

   ls | sort | tee sorted_file_list.tx
   ```

5. `2>` **(Standard Error Redirection):**

- Redirects the standard error (**stderr**) of a command to a file.

```
command 2> error.txt
```

6. `2>>` **(Append Standard Error):**

   - Appends the standard error of a command to a file.

```
command 2>> error.txt
```

7. `&>` **(Redirect Standard Output and Error):**

   - Redirects both standard output and standard error to a file.

```
command &> output_and_error.txt
```

8. `2>&1` **(Merge Standard Error with Standard Output):**

   - Redirects standard error to the same location as standard output.

```
command 2>&1
```

9. `/dev/null` **(Null Device):**

   - Discards data. Useful for suppressing output.

```
command > /dev/null

ls /fake/directory 2> /dev/null
```

10. `tee` **(Split Output):**

    - Reads from standard input and writes to standard output and files simultaneously.

```
command | tee output.txt

ls | tee peanuts.txt banan.txt
```

xargs

`xargs` is a command-line utility in Unix and Unix-like operating systems that builds and executes command lines from standard input. It is often used to process and execute command lines on the output of other commands, making it very powerful for automating repetitive tasks.

Overview of `xargs`

`xargs` takes input from standard input (stdin) or a pipe and passes it as arguments to another command. This is particularly useful for commands that do not accept standard input directly or need arguments to be passed in a specific way.

Basic Syntax

```
xargs [options] [command [initial-arguments]]
```

- **command**: The command to be executed.
- **initial-arguments**: Optional initial arguments for the command.
- **options**: Options to control the behavior of `xargs`.

Common Use Cases

1. **Handling Output from Other Commands**:
    - Combine with commands like `find`, `grep`, `ls`, etc., to process files or output.
2. **Executing Commands with Multiple Arguments**:
    - Run commands with multiple arguments that are generated by another process.

Examples

Example 1: Basic Usage with `echo`

```
echo "file1 file2 file3" | xargs rm
```

This command will delete `file1`, `file2`, and `file3` by passing them as arguments to `rm`.

Example 2: Using `find` with `xargs`

```
find . -name "*.log" | xargs rm
```

This command finds all `.log` files in the current directory and its subdirectories and deletes them.

Example 3: Limiting Arguments per Command

```
find . -type f -print | xargs -n 2 cp -t /backup
```

This command finds all files and copies them to `/backup`, two files at a time (`-n 2`).

Example 4: Handling Spaces and Special Characters

To safely handle filenames with spaces or special characters, use `-0` with `find` and `xargs`.

```
find . -name "*.log" -print0 | xargs -0 rm
```

This command will delete `.log` files, even if their names contain spaces or special characters.

Key Options for `xargs`

- `0, --null`: Use a null character as the delimiter (useful with `find -print0`).
- `n, --max-args=max-args`: Use at most `max-args` arguments per command line.
- `L, --max-lines=max-lines`: Use at most `max-lines` non-blank input lines per command line.

- `I, --replace=replace-str`: Replace occurrences of `replace-str` in the initial arguments with input.
- `P, --max-procs=max-procs`: Run up to `max-procs` processes at a time.
- `t, --verbose`: Print the command line on the standard error output before executing it.
- `p, --interactive`: Prompt the user before running each command.

Advanced Usage

Example 5: Using `I` for Replacement

```
echo "file1 file2 file3" | xargs -I {} mv {} /backup
```

This command moves `file1`, `file2`, and `file3` to the `/backup` directory, using `{}` as a placeholder for each filename.

Example 6: Running Multiple Commands in Parallel

```
cat list_of_urls.txt | xargs -P 4 -n 1 wget
```

This command downloads files from a list of URLs using `wget`, running up to 4 download processes in parallel (`-P 4`).

tar

The `tar` command in Unix and Unix-like operating systems (such as Linux) is used to create, maintain, modify, and extract files from an archive file, commonly called a tarball. The `tar` format is one of the most common archive formats and is often used in conjunction with compression tools such as `gzip` or `bzip2` to create compressed archive files.

Basic Syntax

```
tar [options] [archive-file] [file or directory to be archi
ved]
```

Commonly Used Options

Option	Description
`-c`	Create a new archive.
`-x`	Extract files from an archive.
`-v`	Verbosely list files processed (useful for seeing what's happening).
`-f`	Specify the filename of the archive.
`-z`	Filter the archive through `gzip` for compression or decompression.
`-j`	Filter the archive through `bzip2` for compression or decompression.
`-J`	Filter the archive through `xz` for compression or decompression.
`-t`	List the contents of an archive.
`-r`	Append files to the end of an archive.
`-u`	Only append files newer than the copy in the archive.
`-C`	Change to a directory before performing any operations.
`--exclude`	Exclude files matching a pattern.

Examples

Create a tar Archive

To create a tar archive of a directory, use the `-c` option:

```
tar -cvf archive-name.tar /path/to/directory
```

Create a Compressed tar Archive

To create a tar archive and compress it with `gzip`, use the `-z` option:

```
tar -czvf archive-name.tar.gz /path/to/directory
```

To create a tar archive and compress it with `bzip2`, use the `-j` option:

```
tar -cjvf archive-name.tar.bz2 /path/to/directory
```

To create a tar archive and compress it with `xz`, use the `-J` option:

```
tar -cJvf archive-name.tar.xz /path/to/directory
```

Extract a tar Archive

To extract a tar archive, use the `-x` option:

```
tar -xvf archive-name.tar
```

To extract a compressed tar archive with `gzip`, use the `-z` option:

```
tar -xzvf archive-name.tar.gz
```

To extract a compressed tar archive with `bzip2`, use the `-j` option:

```
tar -xjvf archive-name.tar.bz2
```

To extract a compressed tar archive with `xz`, use the `-J` option:

```
tar -xJvf archive-name.tar.xz
```

List the Contents of a tar Archive

To list the contents of a tar archive without extracting it, use the `-t` option:

```
tar -tvf archive-name.tar
```

Append Files to an Existing tar Archive

To append files to an existing tar archive, use the `-r` option:

```
tar -rvf archive-name.tar file-to-append
```

Exclude Files from a tar Archive

To exclude specific files or directories when creating a tar archive, use the `--exclude` option:

```
tar --exclude='/path/to/exclude' -cvf archive-name.tar /pat
h/to/directory
```

Change Directory When Extracting

To change to a specific directory before extracting, use the `-C` option:

```
tar -xvf archive-name.tar -C /path/to/extract/to
```

6. System Information & Management

System/OS Detail

Command	Description	Example Usage
`uname`	Print system information.	
`-a`	Print all available system information.	`uname -a`
`-s`	Print the kernel name.	`uname -s`
`-r`	Print the kernel release.	`uname -r`
`-v`	Print the kernel version.	`uname -v`
`-m`	Print the machine hardware name.	`uname -m`
`-p`	Print the processor type.	`uname -p`
`-i`	Print the hardware platform.	`uname -i`
`-o`	Print the operating system.	`uname -o`
`lsb_release`	Display Linux Standard Base (LSB) and distribution-specific information.	
`-a`	Print all LSB and distribution information.	`lsb_release -a`
`-d`	Print the description of the distribution.	`lsb_release -d`
`-r`	Print the release number of the distribution.	`lsb_release -r`
`-c`	Print the codename of the distribution.	`lsb_release -c`

`hostnamectl`	Query and change the system hostname and related settings.	
`status`	Display system hostname and related information.	`hostnamectl status`
`set-hostname`	Set a new hostname.	`hostnamectl set-hostname new-hostname`
`cat /etc/os-release`	Display detailed OS information from the `os-release` file.	
		`cat /etc/os-release`
`cat /proc/version`	Display kernel version and other related information.	
		`cat /proc/version`
`dmidecode`	Display hardware information (requires root).	
		`sudo dmidecode`
`lsblk`	List information about block devices.	
		`lsblk`
`df`	Report filesystem disk space usage.	
		`df -h`
`free`	Display memory usage.	
		`free -h`

ACL

Access Control Lists (ACLs) provide a more granular permission model compared to traditional Unix file permissions. They allow you to specify permissions for individual users or groups on files and directories, offering a way to manage complex access requirements.

Overview of ACLs

- **Purpose**: To grant permissions to specific users or groups beyond the standard owner/group/others model.

- **Usage**: Useful for managing file and directory permissions in environments with multiple users and complex access requirements.

Basic ACL Commands

1. `getfacl`

- **Purpose**: Display ACLs for files and directories.
- **Usage**: Shows the current ACLs of a file or directory.

Syntax:

```
getfacl [file/directory]
```

Example:

```
getfacl myfile.txt
```

2. `setfacl`

- **Purpose**: Modify ACLs for files and directories.
- **Usage**: Set or modify ACLs to grant specific permissions.

Syntax:

```
setfacl [options] [acl] [file/directory]
```

Options:

- `m` : Modify existing ACLs or add new ones.
- `x` : Remove ACL entries.
- `b` : Remove all ACL entries (except the base permissions).
- `k` : Remove the default ACL.

Examples:

- **Add a permission for a user**:

  ```
  setfacl -m u:username:rwx myfile.txt
  ```

Grants read, write, and execute permissions to `username` on `myfile.txt`.

- **Add a default permission for a directory**:

  ```
  setfacl -m d:u:username:rx mydir
  ```

  Grants `username` read and execute permissions on all files created in `mydir`.

- **Remove a specific ACL entry**:

  ```
  setfacl -x u:username myfile.txt
  ```

  Removes all ACL entries for `username` on `myfile.txt`.

- **Remove all ACL entries**:

  ```
  setfacl -b myfile.txt
  ```

  Removes all ACL entries, leaving only the base permissions.

3. Viewing ACLs

To view the ACLs associated with a file or directory, you can use `getfacl`:

```
getfacl myfile.txt
```

The output might look like this:

```
# file: myfile.txt
# owner: user
# group: group
user::rw-
user:username:rwx
group::r--
mask::rwx
other::r--
```

- `user::rw-` : Owner's permissions.
- `user:username:rwx` : Permissions for `username`.
- `group::r--` : Group permissions.

- `mask::rwx` : Maximum effective permissions.
- `other::r--` : Permissions for others.

Examples of Use

- **Set ACL to allow a specific user to read and write a file**:

  ```
  setfacl -m u:jane:rw myfile.txt
  ```

- **Set default ACLs on a directory so that new files have read access for a group**:

  ```
  setfacl -m d:g:mygroup:r /mydir
  ```

- **Remove ACL permissions for a specific user**:

  ```
  setfacl -x u:jane myfile.txt
  ```

- **View all ACLs for a directory and its contents**:

  ```
  getfacl -R /mydir
  ```

Devices

In Linux, devices (hardware components like storage drives, network interfaces, etc.) are represented as files within the filesystem, usually located under `/dev`. These device files allow the operating system and applications to interact with hardware through standardized interfaces. There are various types of devices, each represented by a device file in Linux.

Here's a breakdown of the most common device categories and how they are managed in Linux:

1. Block Devices:

- Block devices represent hardware that reads or writes data in fixed-size blocks, such as hard drives, SSDs, and USB drives.

- Block devices are typically found under `/dev` and have names like `/dev/sda`, `/dev/sdb`, `/dev/nvme0n1`, etc.
- You can use the following command to list all block devices:

  ```
  lsblk
  ```

- **Common Block Device Files:**
  - `/dev/sda`, `/dev/sdb`: These represent entire physical disks (e.g., HDDs or SSDs).
  - `/dev/sda1`, `/dev/sdb2`: These represent partitions on the disks.
  - `/dev/nvme0n1`: Represents an NVMe SSD.
- **Commands to Manage Block Devices:**
  - `fdisk -l`: Lists partition tables and block devices.
  - `blkid`: Displays attributes of block devices like UUIDs and filesystem types.
  - `parted`, `gdisk`: Tools to create, modify, and delete partitions.

2. Character Devices:

- Character devices allow for input and output of data one character at a time, such as keyboards, mice, and serial ports.
- Character devices typically interact with the system in real-time, processing data streams.
- Character device files are also located under `/dev`, with names like `/dev/tty`, `/dev/console`, etc.
- **Common Character Device Files:**
  - `/dev/tty`: Represents terminal devices (used for console input/output).
  - `/dev/console`: Represents the system console.
  - `/dev/random` and `/dev/urandom`: Provide access to the kernel's random number generator.
  - `/dev/null`: Discards all data written to it (acts as a "black hole").
  - `/dev/zero`: Provides an endless stream of zero bytes.

- **Commands to Manage Character Devices:**
    - `dmesg`: Shows kernel messages, which often include details about detected hardware.
    - `stty`: Configures and queries terminal line settings.

3. Network Devices:

- Network devices represent interfaces for network communication, such as Ethernet or Wi-Fi cards.
- Network interfaces are not typically represented as files in `/dev`, but you can see them using commands like `ip` or `ifconfig`.
- Examples of network devices:
    - `eth0`, `wlan0`, `enp0s3`: Represent wired and wireless network interfaces.
- **Commands to Manage Network Devices:**
    - `ip link show`: Displays network interfaces.
    - `ifconfig`: Deprecated, but still useful for configuring network interfaces.
    - `iwconfig`: Used for wireless interface management.
    - `ethtool`: Used for displaying and modifying Ethernet device settings.

4. Special Devices:

- These include device files that provide direct access to certain system functions, such as `/dev/null` and `/dev/full`.
- **Special Device Files:**
    - `/dev/null`: Anything written to this device is discarded.
    - `/dev/full`: Acts like a full disk; writing to it will always return a "no space left" error.
    - `/dev/loopX`: Loopback devices used to mount files as if they were block devices.
    - `/dev/fuse`: For user-space filesystem implementations like SSHFS.
    - `/dev/kvm`: Represents Kernel-based Virtual Machine interface, used by virtualization software.

5. Virtual Devices:

- These are not physical devices but rather virtual interfaces created by the operating system for specific purposes.
- **Examples:**
    - `/dev/pts/` : Pseudo-terminal slave devices (used by terminal emulators).
    - `/dev/fb0` : Framebuffer device, used for direct access to display hardware.
    - `/dev/input/` : Represents input devices like keyboards and mice.

6. USB Devices:

- USB devices, such as external drives or peripherals, are represented under `/dev`, typically as block or character devices.
- **Listing USB devices:**

```
lsusb
```

7. Sound Devices:

- Represent audio hardware like sound cards.
- **Common Sound Device Files:**
    - `/dev/snd/` : Contains files for controlling sound devices, including mixers, audio input/output devices.
    - `aplay -l` : Lists sound cards and devices.

8. Disk and Partition Management:

- To see all available disk and partition devices:

```
lsblk
fdisk -l
```

Commands to Interact with Devices:

1. **`lsblk`** :
    - Lists information about all available block devices.

```
lsblk
```

2. `lspci`:

   - Lists information about PCI devices (graphics cards, network cards, etc.).

   ```
   lspci
   ```

3. `lsusb`:

   - Lists information about USB devices connected to the system.

   ```
   lsusb
   ```

4. `dmesg`:

   - Prints kernel messages, often containing information about hardware devices during boot or when connected.

   ```
   dmesg | grep -i usb
   ```

5. `udevadm`:

   - Manages device nodes created by `udev`, the device manager for the Linux kernel.

   ```
   udevadm info --query=all --name=/dev/sda
   ```

Boot

The Linux boot process involves several stages, from powering on the system to loading the operating system and providing a login prompt. Here's a detailed explanation of the steps involved:

1. BIOS/UEFI Initialization:

- **BIOS (Basic Input/Output System)** or **UEFI (Unified Extensible Firmware Interface)** is the firmware that initializes the hardware when the system is powered on.
- It performs the **POST (Power-On Self-Test)**, checking for basic hardware components like CPU, RAM, keyboard, and other devices.
- Once the POST is successful, it looks for a bootable device (hard drive, USB, etc.), usually checking in the boot order configured in the BIOS/UEFI settings.
- The BIOS/UEFI hands over control to the bootloader on the bootable device.

2. Bootloader (GRUB):

- The **bootloader** (most commonly **GRUB** — GNU GRand Unified Bootloader) is responsible for loading the Linux kernel into memory.
- **GRUB** provides a menu that allows users to choose between different kernels or operating systems (if dual-booting).
- The bootloader then loads the selected kernel into memory and passes control to it.

GRUB configuration file is typically located at `/boot/grub/grub.cfg`.

3. Kernel Initialization:

- Once the kernel is loaded into memory, it initializes the hardware components like CPU, memory, and I/O devices.
- The kernel also mounts the **root filesystem** in read-only mode, so it can access the necessary system files.
- The kernel then loads the initial **initramfs** or **initrd** (initial RAM filesystem or disk), which contains necessary drivers and tools required to access the root filesystem and other hardware components, especially for complex hardware setups (e.g., RAID, LVM, etc.).

The kernel is the core of the operating system that manages hardware resources and provides system services.

4. initramfs/initrd:

- **initramfs** (initial RAM filesystem) or **initrd** (initial RAM disk) is a temporary filesystem loaded into memory by the kernel.
- It contains drivers and tools needed to mount the real root filesystem (e.g., RAID, encrypted disks, etc.).
- After mounting the actual root filesystem, control is passed to the init process.
- Once the root filesystem is mounted, `initramfs` or `initrd` is removed from memory.

5. `/sbin/init` (or systemd, Upstart):

- The kernel starts the first user-space process, traditionally `/sbin/init`, but most modern Linux systems use **systemd** or sometimes **Upstart**.
- **systemd** is the most common init system in modern Linux distributions.
- **init/systemd** is the process that manages system initialization and all other processes. It runs with a **PID of 1**.

The role of `init` / `systemd` includes:

- Reading its configuration files (such as `/etc/systemd/system/` in case of systemd).
- Initializing services and daemons as defined (like networking, logging, and various other system processes).

Systemd configuration files are typically found in `/etc/systemd/` and service files in `/etc/systemd/system/`.

6. System Services (Targets or Runlevels):

- System services are started according to the **target** (in systemd) or **runlevel** (in SysVinit) specified.
- **Targets** in systemd replace the traditional **runlevels**. Some common systemd targets include:
    - `multi-user.target`: Multi-user mode without a graphical interface (similar to runlevel 3).
    - `graphical.target`: Multi-user mode with a graphical interface (similar to runlevel 5).

- At this stage, system services like networking, cron jobs, databases, web servers, and other background daemons are started.

You can check the current target using:

```
systemctl get-default
```

To switch targets:

```
systemctl isolate graphical.target
```

7. Login Prompt (TTY or Display Manager):

- Once the system reaches the specified target (multi-user or graphical), the user is presented with a login prompt.
- If running in **console mode** (non-graphical), the system provides a **TTY** login prompt where the user can enter their credentials to access the system.
- If running in **graphical mode**, a **display manager** (like `GDM`, `LightDM`, or `SDDM`) is started to present a graphical login screen where the user can log in to a desktop environment.

8. User Space:

- After login, the user is taken to their shell (in console mode) or desktop environment (in graphical mode).
- At this point, the boot process is complete, and the system is fully operational.

Summary of Steps in Linux Boot Process:

1. **BIOS/UEFI**: Performs hardware checks and hands over control to the bootloader.
2. **Bootloader (GRUB)**: Loads the Linux kernel and passes control to it.
3. **Kernel**: Initializes hardware, mounts root filesystem, and starts `init`.
4. **initramfs/initrd**: Temporary filesystem to load drivers for hardware and mount the real root filesystem.

5. **init/systemd**: The first process (`PID 1`) starts system services based on the target or runlevel.
6. **System Services**: Services like networking, logging, etc., are started.
7. **Login Prompt**: Presents either a console (TTY) or graphical login prompt (via a display manager).
8. **User Space**: After login, the user is in control of the system.

Symlink

What is a Symlink?

A symlink, short for symbolic link, is a type of file in Linux that serves as a reference or pointer to another file or directory. It is a powerful tool for creating shortcuts or aliases to files or directories, making it easier to access and manage files.

Types of Symlinks

There are two types of symlinks:

1. **Soft Symlink**: A soft symlink is a reference to a file or directory that is resolved at runtime. If the original file or directory is deleted or moved, the symlink will break.
2. **Hard Symlink**: A hard symlink is a reference to a file or directory that is resolved at creation time. If the original file or directory is deleted or moved, the hard symlink will still point to the original location.

Creating a Symlink

To create a symlink, use the `ln` command with the `-s` option:

```
ln -s target_file symlink_name
```

- `target_file` is the original file or directory that you want to link to.
- `symlink_name` is the name of the symlink that you want to create.

For example:

```
ln -s /usr/bin/vim /bin/vi
```

This creates a symlink called `vi` in the `/bin` directory that points to the `vim` executable in the `/usr/bin` directory.

Managing Symlinks

Here are some common tasks related to symlinks:

- **List Symlinks**: Use the `ls` command with the `l` option to list symlinks:

```
ls -l
```

- **Check Symlink Target**: Use the `readlink` command to check the target of a symlink:

```
readlink symlink_name
```

- **Remove Symlink**: Use the `rm` command to remove a symlink:

```
rm symlink_name
```

- **Update Symlink**: Use the `ln` command with the `s` option to update the target of a symlink:

```
ln -s new_target_file symlink_name
```

Common Use Cases

Symlinks are useful in various scenarios:

- **Creating shortcuts**: Symlinks can be used to create shortcuts to frequently used files or directories.
- **Versioning**: Symlinks can be used to manage different versions of a file or directory.
- **Dependency management**: Symlinks can be used to manage dependencies between files or directories.

7. Process, Services, & Package Management

Service & Process Management

In Linux, a process is an instance of a running program. Each process is assigned a unique Process ID (PID) and has its own memory space, environment variables, and system resources. Processes can be started, managed, and terminated using various commands and tools available in Linux. Here is a detailed overview of processes in Linux:

Types of Processes

1. **Foreground Processes**:
    - Run directly from the terminal and take user input.
    - Example: `nano` text editor.
2. **Background Processes**:
    - Run without user interaction.
    - Example: A long-running task like a backup script, started with `&`.
3. **Daemon Processes**:
    - Background processes that start at boot and provide services.
    - Example: `httpd` (Apache web server).

Common Process Commands

Command	Description	Example
`ps`	Display information about running processes.	`ps aux`
`top`	Display real-time system summary information and process list.	`top`
`htop`	Interactive process viewer (requires installation).	`htop`
`pgrep`	Search for processes by name.	`pgrep firefox`
`pkill`	Send a signal to processes by name.	`pkill -9 firefox`
`kill`	Send a signal to a process by PID.	`kill -9 1234`

`killall`	Send a signal to all processes by name.	`killall firefox`
`nice`	Start a process with a specified priority.	`nice -n 10 command`
`renice`	Change the priority of an existing process.	`renice -n 10 -p 1234`
`bg`	Resume a suspended job in the background.	`bg %1`
`fg`	Bring a background job to the foreground.	`fg %1`
`jobs`	List active jobs.	`jobs`
`nohup`	Run a command immune to hangups, with output to a non-tty.	`nohup command &`

Process Management Commands

1. `ps`: Display Process Status

```
ps aux        # Display all running processes
ps -ef        # Another format to display all processes
```

2. `top`: Real-Time Process Viewer

```
top           # Interactive process viewer
```

3. `htop`: Enhanced Interactive Process Viewer (requires installation)

```
htop          # Interactive process viewer with more feat
ures
```

4. `pgrep`: Search for Processes

```
pgrep firefox    # Find the PID of all running firefox p
rocesses
```

5. `pkill`: Kill Processes by Name

```
pkill -9 firefox # Forcefully kill all firefox processes
```

6. `kill`: Send a Signal to a Process

```
kill -9 1234     # Forcefully kill the process with PID
1234
```

7. `killall`: Kill Processes by Name

```
killall firefox  # Kill all instances of firefox
```

8. `nice` and `renice`: Set and Adjust Process Priority

```
nice -n 10 command      # Start a command with lower pri
ority
renice -n 10 -p 1234    # Change priority of an existing
process
```

9. `bg` and `fg`: Manage Background and Foreground Jobs

```
bg %1      # Resume job 1 in the background
fg %1      # Bring job 1 to the foreground
```

10. `jobs`: List Active Jobs

```
jobs       # List all active jobs
```

11. `nohup`: Run a Command Immune to Hangups

```
nohup command &  # Run a command that continues running
even after logout
```

Common Signals in Linux:

Each signal has a name (like `SIGKILL`) and a corresponding number (like `9`). Here are some of the most common signals:

Signal	Number	Description	Default Action
`SIGHUP`	1	Hang up detected on controlling terminal or death	Terminates the process

		of controlling process	
`SIGINT`	2	Interrupt from keyboard (Ctrl + C)	Terminates the process
`SIGQUIT`	3	Quit from keyboard (Ctrl + \)	Creates a core dump and terminates
`SIGKILL`	9	Kill signal (cannot be ignored or handled)	Terminates the process immediately
`SIGTERM`	15	Termination signal (default kill signal, can be handled)	Terminates the process
`SIGSTOP`	19	Stop (cannot be ignored)	Stops the process
`SIGCONT`	18	Continue if stopped	Resumes the stopped process
`SIGUSR1`	10	User-defined signal 1	Can be defined by the user/application
`SIGUSR2`	12	User-defined signal 2	Can be defined by the user/application
`SIGALRM`	14	Timer signal from alarm	Terminates the process
`SIGCHLD`	17	Sent to parent when child process terminates	Ignored by default
`SIGSEGV`	11	Segmentation fault	Creates a core dump and terminates

Systemctl

`systemctl` is a command-line utility in Linux used to control the systemd system and service manager. It provides a variety of commands to manage systemd services, check the status of services, enable or disable services to start at boot, and more. Here's a detailed overview of its functionality:

Basic Concept

- **Systemd**: Systemd is a system and service manager for Linux operating systems. It is responsible for initializing the system (starting the user space and managing services).

- **Unit Files**: Systemd uses unit files to represent services, mount points, devices, sockets, and other resources that systemd manages.

Common `systemctl` Commands

Service Management

Command	Description	Example
`systemctl start <service>`	Start a service	`systemctl start httpd`
`systemctl stop <service>`	Stop a service	`systemctl stop httpd`
`systemctl restart <service>`	Restart a service	`systemctl restart httpd`
`systemctl reload <service>`	Reload the configuration of a service without restarting it	`systemctl reload httpd`
`systemctl status <service>`	Check the status of a service	`systemctl status httpd`
`systemctl enable <service>`	Enable a service to start at boot	`systemctl enable httpd`
`systemctl disable <service>`	Disable a service from starting at boot	`systemctl disable httpd`
`systemctl is-active <service>`	Check if a service is active	`systemctl is-active httpd`
`systemctl is-enabled <service>`	Check if a service is enabled at boot	`systemctl is-enabled httpd`
`systemctl daemon-reload`	Reload systemd manager configuration	`systemctl daemon-reload`

System Management

Command	Description	Example
`systemctl reboot`	Reboot the system	`systemctl reboot`
`systemctl poweroff`	Power off the system	`systemctl poweroff`
`systemctl suspend`	Suspend the system	`systemctl suspend`

`systemctl hibernate`	Hibernate the system	`systemctl hibernate`
`systemctl halt`	Halt the system	`systemctl halt`
`systemctl isolate <target>`	Change the system state to the specified target	`systemctl isolate multi-user.target`

Cron jobs

Cron jobs in Linux are used for automating tasks that you want to run at specific intervals. Cron is a time-based job scheduler that enables you to run commands or scripts automatically at a particular time or date. It is especially useful for repetitive tasks like backups, monitoring, or updating systems.

Key Components of Cron:

1. **Cron Daemon (`crond`)**:
    - A background service that schedules and executes the commands specified in the cron jobs. It constantly runs in the background to check and execute scheduled tasks.
2. **Cron Table (Crontab)**:
    - A configuration file where you define cron jobs. Each user (including the system itself) can have their own crontab file. You can edit this file to schedule your jobs.
3. **Cron Format**:
   Each line in the crontab follows this syntax:

```
* * * * * command-to-be-executed
| | | | |
| | | | +---- Day of the week (0-7) (Sunday to Saturday,
Sunday can be both 0 and 7)
| | | +------ Month (1-12)
| | +-------- Day of the month (1-31)
| +---------- Hour (0-23)
+------------ Minute (0-59)
```

Example:

```
30 2 * * 1 /home/user/backup.sh
```

This cron job runs every Monday at 2:30 AM and executes the script `backup.sh` located in `/home/user/`.

Symbols in Cron:

- — Matches any value in that field.
- `,` — Allows you to specify multiple values.
  - Example: `1,2,3` would match the 1st, 2nd, and 3rd of the month.
- — Specifies a range of values.
  - Example: `1-5` means from the 1st to the 5th.
- `/` — Specifies step values.
  - Example: `/5` in the minute field means every 5 minutes.

Managing Cron Jobs:

1. **List Cron Jobs**:
   To list your cron jobs:

   ```
   crontab -l
   ```

2. **Edit Cron Jobs**:
   To edit the cron jobs:

   ```
   crontab -e
   ```

   This opens your crontab file in the default text editor (usually `vim` or `nano`).

3. **Remove Cron Jobs**:
   To remove all cron jobs for the current user:

   ```
   crontab -r
   ```

Special Strings in Cron:

There are some predefined cron job schedules you can use instead of the numeric format:

- `@reboot` — Runs once at system startup.
- `@daily` — Runs once a day (equivalent to `0 0 * * *`).
- `@weekly` — Runs once a week (equivalent to `0 0 * * 0`).
- `@monthly` — Runs once a month (equivalent to `0 0 1 * *`).
- `@yearly` — Runs once a year (equivalent to `0 0 1 1 *`).

Examples of Common Cron Jobs:

1. **Backup a directory every day at midnight**:

   ```
   0 0 * * * /usr/bin/rsync -a /home/user/docs /home/user/backup/
   ```

2. **Clear cache every Sunday at 3 AM**:

   ```
   0 3 * * 0 /usr/bin/clean_cache.sh
   ```

3. **Run a script every 15 minutes**:

   ```
   */15 * * * * /home/user/myscript.sh
   ```

Cron Permissions:

- **/etc/cron.allow** — Defines which users are allowed to use cron jobs.
- **/etc/cron.deny** — Defines which users are denied access to cron jobs.

If neither of these files exists, cron is available for all users by default.

Logs:

To check logs of cron jobs, you can view the system log file where cron messages are stored:

```
cat /var/log/syslog | grep cron
```

Conclusion:

Cron jobs are essential for automating routine tasks, and mastering them can significantly improve your productivity in managing systems. You can schedule backups, monitor services, update systems, or perform any other repetitive task efficiently with cron.

dpkg (Debian Package Management)

`dpkg` is a command-line tool in Linux used for managing packages. It is primarily used in Debian-based distributions like Debian and Ubuntu. `dpkg` allows users to install, remove, and manage software packages on their system.

Here are some common uses of `dpkg`:

1. **Install a Package:**
   To install a package using
   `dpkg`, you can use the `i` option followed by the path to the package file. For example:

   ```
   sudo dpkg -i package.deb
   ```

2. **Remove a Package:**
   To remove a package, you can use the
   `r` option followed by the package name. For example:

   ```
   sudo dpkg -r package_name
   ```

3. **List Installed Packages:**
   You can list all installed packages using the
   `l` option:

   ```
   dpkg -l
   ```

4. **Query Information About a Package:**
   To get information about a specific package, you can use the
   `I` option followed by the package name. For example:

```
dpkg -I package_name
```

5. **Reconfigure a Package:**
   If you need to reconfigure a package after installation, you can use the `reconfigure` option followed by the package name. For example:

   ```
   sudo dpkg-reconfigure package_name
   ```

6. **List Files Installed by a Package:**
   To list all files installed by a package, you can use the `L` option followed by the package name. For example:

   ```
   dpkg -L package_name
   ```

7. **Fix Broken Dependencies:**
   If there are broken dependencies, you can attempt to fix them using the `-configure` option:

   ```
   sudo dpkg --configure -a
   ```

8. **To get the total number of packages installed :**

   ```
   dpkg -l | grep '^ii' | wc -l
   ```

   - **`dpkg -l`**: This command lists all installed packages on the system along with their details.
   - **`grep '^ii'`**: This command filters the output of **`dpkg -l`** to include only lines that start with **`ii`**, which indicates installed packages. The **`^ii`** pattern matches lines where the second field starts with **`ii`**.
   - **`wc -l`**: This command counts the number of lines in the filtered output, which corresponds to the number of installed packages.

Curl

`curl` is a command-line tool used to transfer data to or from a server using various protocols, including HTTP, HTTPS, FTP, and many more. It is commonly used to make web requests, download files, and interact with APIs. Below is a detailed overview of `curl`, its usage, and some common examples.

Basic Usage

The basic syntax for `curl` is:

```
curl [options] [URL]
```

Common Options and Examples

1. Fetching a Web Page

To fetch the content of a web page:

```
curl <http://example.com>
```

2. Saving the Output to a File

To save the output to a file instead of displaying it:

```
curl -o output.html <http://example.com>
```

3. Follow Redirects

By default, `curl` does not follow redirects. Use the `-L` option to enable this: This command will start by making a request to `http://example.com`. If the server responds with a redirect (e.g., `301 Moved Permanently` or `302 Found`), `curl` will automatically follow the redirect and make a new request to the URL

```
curl -L <http://example.com>

curl -L --max-redirs 10 http://example.com
```

```
# By default, curl will follow up to 50 redirects. You can
change this limit with the --max-redirs option
```

4. Fetching HTTP Headers

To fetch and display only the HTTP headers of a response:

```
curl -I <http://example.com>
```

5. Sending Data with POST Method

To send data to a server using the POST method:

```
curl -X POST -d "key1=value1&key2=value2" <http://example.c
om/form>
```

6. Sending JSON Data

To send JSON data in a POST request:

```
curl -X POST -H "Content-Type: application/json" -d '{"key
1":"value1","key2":"value2"}' <http://example.com/api>
```

7. Downloading a File

To download a file from a URL:

```
curl -O <http://example.com/file.zip>
```

8. Using Basic Authentication

To make a request with basic authentication:

```
curl -u username:password <http://example.com>
```

9. Custom HTTP Headers

To include custom HTTP headers in a request:

```
curl -H "Authorization: Bearer token" <http://example.com/a
pi>
```

10. Limiting the Download Rate

To limit the download rate:

```
curl --limit-rate 100k <http://example.com/file.zip>
```

Additional Resource

This is contains useful resources to learn further and practice Hands- on Linux commands

Linux for Everyone: The only resource you Need.
What is Linux & Why you should Learn?
https://medium.com/@rajveer_0101/the-ultimate-guide-to-linux-mastery-c691c09b437c

8. Networking & Security

Networking

There are various networking tools which can be used to analyze the network

1. **ping**
2. **traceroute**
3. **nslookup** and **dig**
4. **host**
5. **netstat** and **ss**

6. **lsof**
7. **nmap**
8. **curl** and **wget**
9. **telnet** and **nc** (netcat)
10. **tcpdump**
11. **iftop** and **nload**
12. **route** and **ip route**
13. **arp**
14. **ifconfig** and **ip addr**
15. **route** and **ip route**

1. ping

Description: Sends ICMP ECHO_REQUEST packets to network hosts to test reachability and measure round-trip time.

Common options:

- `c <count>`: Send only the specified number of packets.
- `i <interface>`: Use the specified network interface.
- `w <timeout>`: Wait for the specified number of seconds for a response.
- `W <wait>`: Wait for the specified number of seconds for a response, then stop sending packets.
- `s <packet_size>`: Send packets of the specified size.
- `f`: Flood ping packets to the target.

```
ping -c 4 -i eth0 -w 2 example.com
```

2. traceroute

Description: Traces the route to a network host by sending packets with increasing Time-To-Live (TTL) values and analyzing the ICMP TIME_EXCEEDED messages returned by intermediate routers.

Common options:

- `m <max_ttl>`: Set the maximum TTL value.

- `n`: Display numerical addresses instead of hostnames.
- `w <wait>`: Wait for the specified number of seconds for a response.
- `I <interface>`: Use the specified network interface.

```
traceroute -m 10 -n -w 2 example.com
```

3. **nslookup** and **dig**

Description: Query the Domain Name System (DNS) to obtain information about a specific domain name or IP address. `nslookup` is a simple command-line tool, while `dig` (Domain Information Groper) is more powerful and flexible.

Common options for dig:

- `@<nameserver>`: Use the specified nameserver for the query.
- `+short`: Display only the answers, without the header or additional information.
- `+trace`: Perform a DNS trace, showing the sequence of DNS servers used to resolve the query.
- `+stats`: Display statistics about the query.

```
dig +short +trace example.com
```

4. **host**

Description: A simple command-line tool that performs a DNS lookup and displays the IP addresses associated with a given hostname.

Common options:

- `a`: Display all IP addresses associated with the hostname.
- `t <type>`: Specify the record type to look up (e.g., A, AAAA, MX, etc.).

```
host -a example.com
```

5. **netstat** and **ss**

Description: Display network connections, routing tables, interface statistics, and other network-related information. `netstat` is an older command, while `ss` is its modern replacement.

Common options for ss:

- `t`: Display TCP connections.
- `u`: Display UDP connections.
- `n`: Display numerical addresses instead of hostnames.
- `p`: Display process information associated with each connection.
- `i`: Display interface statistics.

```
ss -tunp
```

6. lsof

Description: Lists open files and the processes that opened them. This can be useful for identifying network connections and the processes associated with them.

Common options:

- `i`: Display information about network files (e.g., sockets, pipes, and FIFOs).
- `n`: Display numerical addresses instead of hostnames.
- `p`: Display the PID and name of the process associated with each file.

```
lsof -i -n -p
```

7. nmap

Description: A powerful network mapping and port scanning tool used to identify hosts and services on a target network. Nmap uses raw IP packets to determine the hosts and services available on a target network, as well as the operating systems and types of packet filters/firewalls in use.

Common options:

- `F`: Use the "fast" scan option, which uses a small number of packets and is faster

```
nmap -F 192.168.1.0/24
```

8. curl and **wget**

Description: Command-line tools for making HTTP requests and transferring files from or to a server. Both tools can be used to test web application functionality, retrieve files, or perform simple web application attacks.

Common options for curl:

- `X <method>`: Use the specified HTTP method (e.g., GET, POST, PUT, DELETE, etc.).
- `d <data>`: Send the specified data in the request body (for POST or PUT requests).
- `H <header>`: Add the specified header to the request.
- `c <cookie>`: Include the specified cookie in the request.
- `o <output>`: Write the output to the specified file.

```
curl -X POST -d 'username=admin&password=secret' -H 'Conten
t-Type: application/x-www-form-urlencoded' <http://example.
com/login>
```

Common options for wget:

- `X <method>`: Use the specified HTTP method (e.g., GET, POST, PUT, DELETE, etc.).
- `d <data>`: Send the specified data in the request body (for POST or PUT requests).
- `H <header>`: Add the specified header to the request.
- `c <cookie>`: Include the specified cookie in the request.
- `O <output>`: Write the output to the specified file.

```
wget --post-data 'username=admin&password=secret' --header
'Content-Type: application/x-www-form-urlencoded' <http://e
xample.com/login>
```

9. **telnet** and **nc** (netcat)

Description: Telnet is a simple command-line tool for establishing a TCP connection to a remote host and interacting with it using plaintext. Netcat (nc) is a more powerful and flexible alternative to telnet, supporting both TCP and UDP connections, as well as various options for sending and receiving data.

Common options for nc:

- `zv`: Scan for listening daemons (ports with services) on the target host.
- `u`: Use UDP instead of TCP.
- `w <timeout>`: Set the timeout for connections and data transfers.
- `n`: Don't resolve hostnames.
- `z`: Scan for listening daemons, without sending any data.

```
nc -zv 192.168.1.100
```

10. tcpdump

Description: A powerful command-line tool for capturing and analyzing network traffic in real-time. Tcpdump can be used to monitor network activity, troubleshoot network issues, and analyze network protocols.

Common options:

- `i <interface>`: Capture traffic on the specified network interface.
- `n`: Don't resolve hostnames.
- `nn`: Don't resolve hostnames or port numbers.
- `w <file>`: Write the packet data to the specified file.
- `r <file>`: Read packet data from the specified file.
- `x`: Print the contents of packets in both ASCII and hex.
- `xx`: Print the contents of packets in hex only.

```
tcpdump -i eth0 -n -w capture.pcap
```

11. **iftop** and **nload**

Description: `iftop` is a network traffic monitor similar to the top command, displaying a table of current network usage by host and port. `nload` is a console application which monitors network traffic and bandwidth usage in real-time.

Common options for iftop:

- `i <interface>`: Monitor traffic on the specified network interface.
- `n`: Don't resolve hostnames.

- `m`: Display traffic in megabits per second (Mbps).

```
iftop -i eth0 -n -m
```

12. route and **ip route**

Description: Display or manipulate the kernel routing table. `route` is an older command, while `ip route` is its modern replacement.

Common options for ip route:

- `show`: Display the kernel routing table.
- `add <destination>/<mask> via <gateway> dev <interface>`: Add a new route to the kernel routing table.
- `delete <destination>/<mask>`: Delete a route from the kernel routing table.

```
ip route show
```

13. arp

Description: Manipulate the ARP table, used by the Linux kernel to map IP addresses to MAC addresses.

Common options:

- `a`: Display the ARP table.
- `s <ip> <mac>`: Add an entry to**14. ifconfig** and **ip addr**

Description: Display or configure network interfaces. `ifconfig` is an older command, while `ip addr` is its modern replacement.

Common options for ip addr:

- `show`: Display information about network interfaces and their IP addresses.
- `add <address>/<prefix> dev <interface>`: Add an IP address to a network interface.
- `delete <address>/<prefix> dev <interface>`: Delete an IP address from a network interface.

```
ip addr show
```

15. route and **ip route**

Description: Display or manipulate the kernel routing table. `route` is an older command, while `ip route` is its modern replacement.

Common options for ip route:

- `show`: Display the kernel routing table.
- `add <destination>/<mask> via <gateway> dev <interface>`: Add a new route to the kernel routing table.
- `delete <destination>/<mask>`: Delete a route from the kernel routing table

```
ip route show
```

SSH

SSH (Secure Shell)

SSH (Secure Shell) is a protocol used to securely connect to a remote machine or server over a network. It encrypts the connection, ensuring that data is securely transmitted between the client and the server.

Basic SSH Command

To connect to a remote server using SSH, you use the following command:

```
ssh username@remote_host
```

- **`username`**: The username on the remote machine.
- **`remote_host`**: The IP address or domain name of the remote machine.

Example

```
ssh user@example.com
```

This command will prompt you for the password of the user on the remote machine. Once authenticated, you'll be connected to the remote machine's terminal.

SSH with a Specific Port

If the SSH server is running on a port other than the default port 22, you can specify the port number using the `-p` option:

```
ssh -p port_number username@remote_host
```

Example:

```
ssh -p 2222 user@example.com
```

In this example, SSH connects to the server on port 2222 instead of the default port 22.

SSH Key-Based Authentication

For enhanced security, you can use SSH key-based authentication instead of passwords:

1. **Generate SSH Key Pair:**

   ```
   ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
   ```

   This generates a pair of keys (private and public). By default, they are stored in `~/.ssh/id_rsa` (private) and `~/.ssh/id_rsa.pub` (public).

2. **Copy the Public Key to the Remote Server:**

   ```
   ssh-copy-id username@remote_host
   ```

   This copies your public key to the remote server, allowing you to log in without a password.

3. **Log in Using SSH Key:**

   ```
   ssh username@remote_host
   ```

Now, SSH will use your private key for authentication.

Common SSH Options

- `i path_to_key`: Specify a private key file for authentication.

```
ssh -i ~/.ssh/id_rsa username@remote_host
```

- `L`: Forward a local port to a remote machine.

  ```
  ssh -L local_port:remote_host:remote_port username@remot
  e_host
  ```

- `R`: Forward a remote port to a local machine.

  ```
  ssh -R remote_port:localhost:local_port username@remote_
  host
  ```

Network Sharing

1. NFS (Network File System)

How it works:

NFS allows a client to access files on a remote server as if they were local. The client sends a request to the server to mount a shared directory, and the server responds with the file system information. The client can then access the files in the shared directory.

Step-by-Step Process:

Server Side:

1. **Install NFS Server**:

   ```
   sudo apt install nfs-kernel-server
   ```

2. **Create a shared directory**:

   ```
   sudo mkdir /shared/directory
   ```

3. **Set permissions**:

```
sudo chmod 755 /shared/directory
```

4. **Edit /etc/exports**:
   Add the following line to share the directory with the client IP address: Replace
   `192.168.1.100` with the client's IP address.

   ```
   /shared/directory 192.168.1.100(ro,async,no_subtree_check)
   ```

5. **Restart NFS Service**:

   ```
   sudo systemctl restart nfs-kernel-server
   ```

Client Side:

1. **Install NFS Client**:

   ```
   sudo apt install nfs-common
   ```

2. **Create a mount point**:

   ```
   sudo mkdir /mnt/nfs
   ```

3. **Mount the shared directory**:
   Replace
   `server_ip` with the server's IP address.

   ```
   sudo mount -t nfs4 server_ip:/shared/directory /mnt/nfs
   ```

4. **Verify the mount**:

   ```
   df -h
   ```

File Transfer:

1. **Copy files to the shared directory**:

```
cp /local/file /mnt/nfs/
```

2. **Verify the file transfer**:

```
ls /mnt/nfs/
```

2. Samba

How it works:

Samba provides a way to share files and printers between Linux and Windows systems. The client sends a request to the server to access a shared resource, and the server responds with the resource information.

Step-by-Step Process:

Server Side:

1. **Install Samba**:

```
sudo apt install samba
```

2. **Create a shared directory**:

```
sudo mkdir /shared/directory
```

3. **Set permissions**:

```
sudo chmod 755 /shared/directory
```

4. **Edit /etc/samba/smb.conf**:
   Add the following lines to share the directory:
   Replace
   `username` with the username of the user who will access the share.

```
[shared]
path = /shared/directory
available = yes
valid users = username
read only = no
```

```
browsable = yes
public = yes
writable = yes
```

5. **Restart Samba Service**:

   ```
   sudo systemctl restart smbd
   ```

Client Side (Windows):

1. **Open File Explorer**:
   Navigate to
   `\\\\server_ip\\shared`.
2. **Enter credentials**:
   Enter the username and password of the user who has access to the share.
3. **Access the shared directory**:
   The shared directory will be displayed in the File Explorer.

File Transfer:

1. **Copy files to the shared directory**:
   Right-click and select "Copy" or use the "Copy" button.
2. **Verify the file transfer**:
   Refresh the File Explorer to see the copied files.

3. FTP (File Transfer Protocol)

How it works:

FTP allows users to transfer files between systems over a network. The client sends a request to the server to access a file, and the server responds with the file information.

Step-by-Step Process:

Server Side:

1. **Install FTP Server**:

   ```
   sudo apt install vsftpd
   ```

2. **Create a shared directory**:

```
sudo mkdir /shared/directory
```

3. **Set permissions**:

   ```
   sudo chmod 755 /shared/directory
   ```

4. **Edit /etc/vsftpd.conf**:
   Add the following lines to configure the FTP server:

   ```
   anonymous_enable=NO
   local_enable=YES
   write_enable=YES
   ```

5. **Restart FTP Service**:

   ```
   sudo systemctl restart vsftpd
   ```

Client Side:

1. **Install FTP Client**:

   ```
   sudo apt install file**Client Side:**
   ```

2. **Install FTP Client** (if not already installed):

   ```
   sudo apt install ftp
   ```

   Alternatively, you can use a graphical client like FileZilla.

3. **Connect to the FTP Server**:

   ```
   ftp server_ip
   ```

   Replace `server_ip` with the server's IP address.

4. **Login**:
   Enter the username and password when prompted.

5. **Navigate to the Shared Directory**:
   Use the
   `cd` command to change to the shared directory:

   ```
   cd /shared/directory
   ```

File Transfer:

1. **Upload a File**:
   Use the
   `put` command to upload a file:

   ```
   put localfile.txt
   ```

2. **Download a File**:
   Use the
   `get` command to download a file:

   ```
   get remotefile.txt
   ```

3. **List Files**:
   Use the
   `ls` command to list files in the current directory:

   ```
   ls
   ```

4. SSH (Secure Shell)

How it works:

SSH allows secure remote access and file transfer over an encrypted connection. SCP (Secure Copy Protocol) and SFTP (SSH File Transfer Protocol) are commonly used for file transfers.

Step-by-Step Process:

Server Side:

1. **Install OpenSSH Server**:

   ```
   sudo apt install openssh-server
   ```

2. **Start SSH Service**:

   ```
   sudo systemctl start ssh
   ```

3. **Enable SSH to Start on Boot**:

   ```
   sudo systemctl enable ssh
   ```

Client Side:

1. **Connect to the SSH Server**:

   ```
   ssh username@server_ip
   ```

   Replace `username` with your server username and `server_ip` with the server's IP address.

2. **Authenticate**:
   Enter the password when prompted.

File Transfer:

1. **Using SCP**:

   - **Upload a File**:

     ```
     scp localfile.txt username@server_ip:/remote/path
     ```

   - **Download a File**:

     ```
     scp username@server_ip:/remote/path/remotefile.txt /l
     ocal/path
     ```

2. **Using SFTP**:

   - Start SFTP session:

     ```
     sftp username@server_ip
     ```

   - **Upload a File**:

```
put localfile.txt
```

- **Download a File**:

```
get remotefile.txt
```

- **List Files**:

```
ls
```

5. Using rsync

How it works:

`rsync` is a powerful tool for syncing files and directories between systems. It uses a delta-transfer algorithm, which means it only transfers the changes made to files, making it efficient.

Step-by-Step Process:

1. **Install rsync** (if not already installed):

```
sudo apt install rsync
```

2. **Sync Files**:

- **From Local to Remote**:

```
rsync -avz /local/path username@server_ip:/remote/pat
h
```

- **From Remote to Local**:

```
rsync -avz username@server_ip:/remote/path /local/pat
h
```

3. **Verify Sync**:
Use the
`ls` command to verify that files have been transferred successfully.

Conclusion

By following the above steps for each method, you can successfully transfer and receive files over a network in a Linux environment. Each method has its strengths, so choose the one that best fits your needs based on the environment and requirements.

Http server

To start a simple HTTP server in Linux, you can use the `python` or `http-server` command. Below are examples for both methods:

Using Python:

1. **Navigate to the directory you want to serve:**

   ```
   cd /path/to/directory
   ```

2. **Start the Python HTTP server:**

   ```
   python3 -m http.server 8080
   ```

   This command starts a basic HTTP server on port 8000 (you can specify a different port if needed).

3. **Access the server in a web browser:**
   Open a web browser and go to `http://localhost:8000`. You should see the contents of the directory served.

Using Node.js (http-server package):

1. **Install the `http-server` package:**

   ```
   npm install -g http-server
   ```

2. **Navigate to the directory you want to serve:**

   ```
   cd /path/to/directory
   ```

3. **Start the HTTP server:**

   ```
   http-server -p 8080
   ```