Vigneswaraya namaha

Linux

Linux Operating System

Linux is an open-source operating system based on the Unix architecture, widely recognized for its versatility, security, and stability. Originally developed by Linus Torvalds in 1991, Linux has since evolved into a collaborative project supported by a large community of developers and users.

Key Features of Linux

1. **Open Source**: Linux is freely available and its source code can be modified, allowing users to customize the system to their needs. This fosters innovation and community collaboration.
2. **Stability and Reliability**: Linux systems are known for their robustness and can run for long periods without requiring a reboot. This makes it an ideal choice for servers and critical applications.
3. **Security**: Linux has a strong security model and is less prone to viruses and malware compared to other operating systems. Its permission and user role features enhance system security.
4. **Variety of Distributions**: There are many Linux distributions (distros), such as Ubuntu, Fedora, CentOS, and Debian, catering to different user needs, from beginners to advanced users.
5. **Community Support**: A large and active community contributes to continuous improvement and support. Users can find extensive documentation, forums, and online resources.
6. **Compatibility**: Linux supports a wide range of hardware platforms, from personal computers to mainframes, and is compatible with many software applications.
7. **Customization**: Users have the freedom to modify the operating system as needed. From the kernel to the graphical user interface, Linux can be tailored to meet specific requirements.
8. **Command-Line Interface**: While many distributions offer graphical interfaces, Linux provides powerful command-line tools that allow users to perform complex tasks efficiently.

Why Linux is Popular

- **Server Environments**: Linux dominates the server market due to its reliability, security, and scalability. Major web servers, database servers, and cloud infrastructures run on Linux.
- **Development Platform**: Many developers prefer Linux for programming and software development due to its powerful tools and flexibility. It is the preferred OS for web development and data science.
- **Cost-Effective**: Being open-source and free to use, Linux reduces software licensing costs, making it an attractive option for individuals and businesses alike.
- **Use in Embedded Systems**: Linux powers many embedded systems and devices, including smartphones (Android), routers, and IoT devices, leading to its widespread adoption.

Linux Basic Commands

Command	Description	Example
`ls`	Lists files and directories in the current directory.	`ls`
`cd`	Changes the current directory.	`cd /home/user`
`pwd`	Prints the current working directory.	`pwd`
`mkdir`	Creates a new directory.	`mkdir new_folder`
`rmdir`	Removes an empty directory.	`rmdir old_folder`
`rm`	Removes files or directories.	`rm file.txt` `rm -r folder_name` (for directories)
`cp`	Copies files or directories.	`cp source.txt destination.txt`

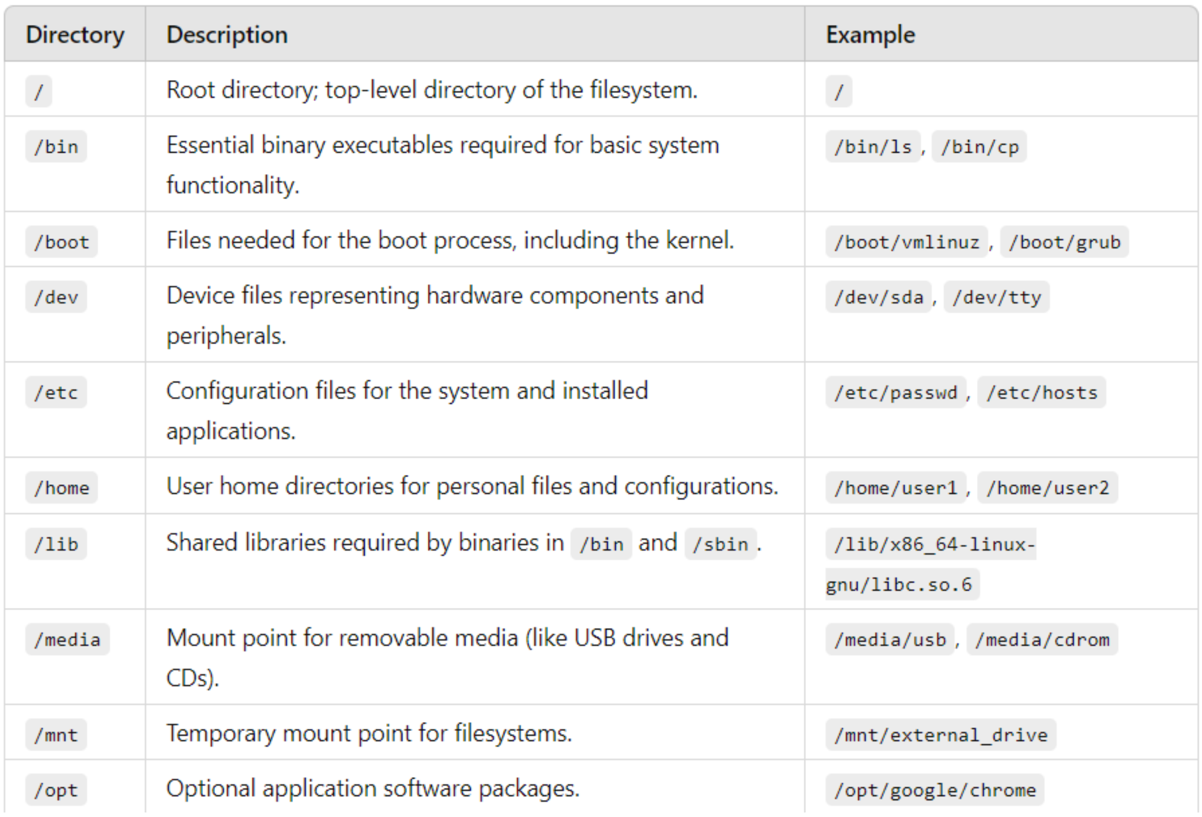
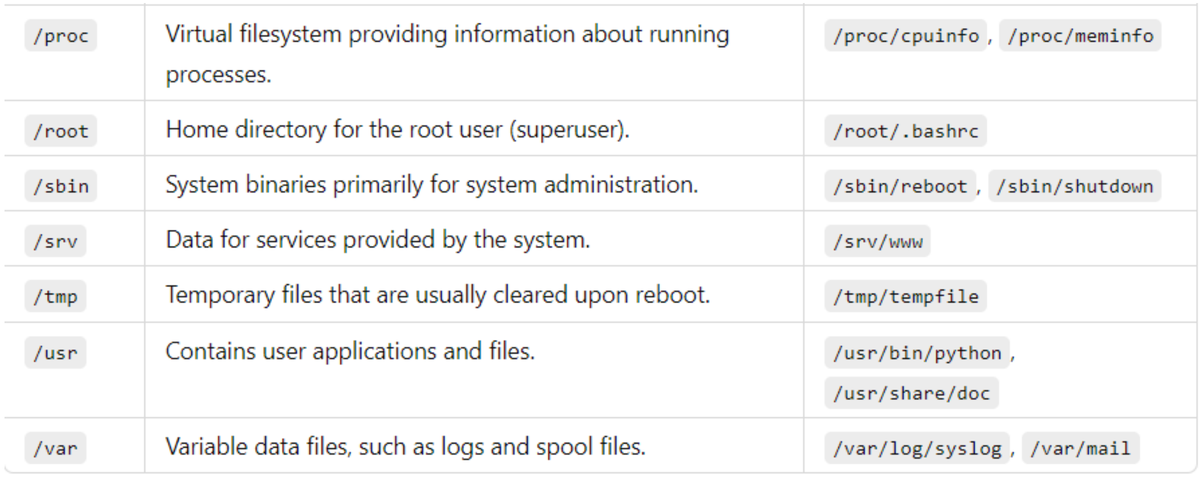
`mv`	Moves or renames files or directories.	`mv oldname.txt newname.txt`
		`mv file.txt /home/user/`

`cat`	Concatenates and displays file content.	`cat file.txt`
`less`	Views the content of a file one page at a time.	`less file.txt`
`touch`	Creates a new empty file or updates the timestamp of an existing file.	`touch newfile.txt`
`echo`	Displays a line of text or variable value.	`echo "Hello, World!"`
`man`	Displays the manual for a command.	`man ls`
`chmod`	Changes the permissions of a file or directory.	`chmod 755 script.sh`
`chown`	Changes the owner of a file or directory.	`chown user:group file.txt`
`find`	Searches for files and directories.	`find /path/to/search -name "*.txt"`
`grep`	Searches for specific text within files.	`grep "search_term" file.txt`
`df`	Displays disk space usage for all mounted filesystems.	`df -h`
`du`	Displays disk usage of files and directories.	`du -h /path/to/directory`
`top`	Displays real-time system processes and resource usage.	`top`
`ps`	Displays currently running processes.	`ps aux`
`kill`	Terminates a process by its process ID.	`kill 1234`

Linux Directory Hierarchy

In Linux, the directory hierarchy is structured in a tree-like format starting from the root directory, denoted by /. Here's an overview of the standard directory structure:

Common Directories Under /

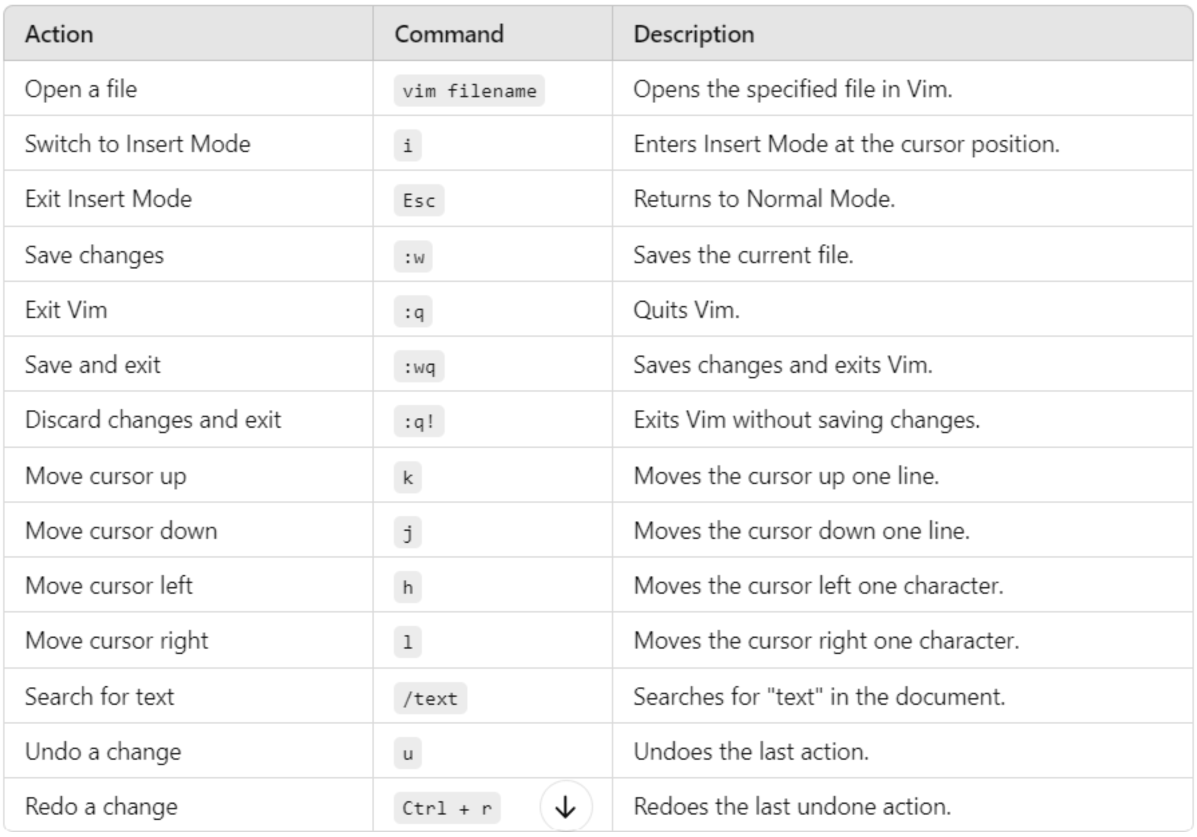
Directory	Description	Example
`/`	Root directory; top-level directory of the filesystem.	`/`
`/bin`	Essential binary executables required for basic system functionality.	`/bin/ls`, `/bin/cp`
`/boot`	Files needed for the boot process, including the kernel.	`/boot/vmlinuz`, `/boot/grub`
`/dev`	Device files representing hardware components and peripherals.	`/dev/sda`, `/dev/tty`
`/etc`	Configuration files for the system and installed applications.	`/etc/passwd`, `/etc/hosts`
`/home`	User home directories for personal files and configurations.	`/home/user1`, `/home/user2`
`/lib`	Shared libraries required by binaries in `/bin` and `/sbin`.	`/lib/x86_64-linux-gnu/libc.so.6`
`/media`	Mount point for removable media (like USB drives and CDs).	`/media/usb`, `/media/cdrom`
`/mnt`	Temporary mount point for filesystems.	`/mnt/external_drive`
`/opt`	Optional application software packages.	`/opt/google/chrome`
`/proc`	Virtual filesystem providing information about running processes.	`/proc/cpuinfo`, `/proc/meminfo`
`/root`	Home directory for the root user (superuser).	`/root/.bashrc`
`/sbin`	System binaries primarily for system administration.	`/sbin/reboot`, `/sbin/shutdown`
`/srv`	Data for services provided by the system.	`/srv/www`
`/tmp`	Temporary files that are usually cleared upon reboot.	`/tmp/tempfile`
`/usr`	Contains user applications and files.	`/usr/bin/python`, `/usr/share/doc`
`/var`	Variable data files, such as logs and spool files.	`/var/log/syslog`, `/var/mail`

Vim Editor

Vim (Vi IMproved) is a highly configurable and powerful text editor used primarily in command-line interfaces. It is an enhanced version of the original vi editor, offering more features, including syntax highlighting, multi-level undo, and a wide range of plugins. Vim operates in different modes, primarily Normal Mode, Insert Mode, and Command Mode.

Vim is a powerful text editor that operates in multiple modes, allowing for efficient text manipulation. Its extensive command set provides flexibility for editing tasks, making it a popular choice among programmers and system administrators.

- Normal Mode: The default mode where you can navigate and manipulate text.
- Insert Mode: Used for inserting text; accessed by pressing i.
- Command Mode: Used for executing commands; accessed by pressing :.



Action	Command	Description
Open a file	`vim filename`	Opens the specified file in Vim.
Switch to Insert Mode	`i`	Enters Insert Mode at the cursor position.
Exit Insert Mode	`Esc`	Returns to Normal Mode.
Save changes	`:w`	Saves the current file.
Exit Vim	`:q`	Quits Vim.
Save and exit	`:wq`	Saves changes and exits Vim.
Discard changes and exit	`:q!`	Exits Vim without saving changes.
Move cursor up	`k`	Moves the cursor up one line.
Move cursor down	`j`	Moves the cursor down one line.
Move cursor left	`h`	Moves the cursor left one character.
Move cursor right	`l`	Moves the cursor right one character.
Search for text	`/text`	Searches for "text" in the document.
Undo a change	`u`	Undoes the last action.
Redo a change	`Ctrl + r`	Redoes the last undone action.

Command path

In Linux, the "command path" refers to the location of executable files for commands that you can run in the terminal. The command which python is used to find the path of the Python interpreter that will be executed when you type python in the terminal. It helps you identify the location of the Python executable that is currently set in your system's PATH. Example below

```
$which python
Output: /usr/bin/python
$which python3
Output: /usr/bin/python3
```

Determine the type of command

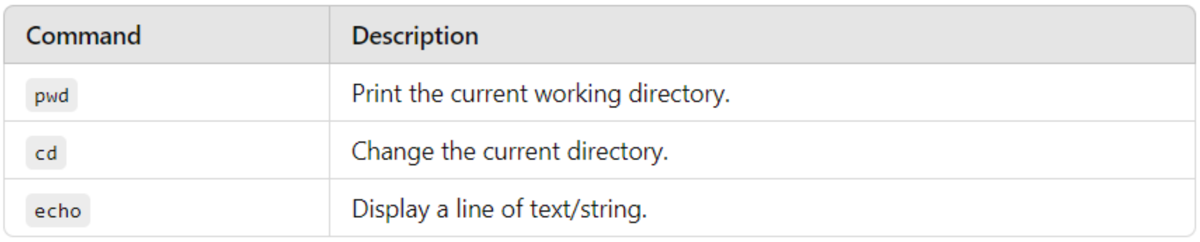
1.Builtin Commands

Built-in commands are integrated into the shell itself. They are executed directly by the shell, making them faster and less resource-intensive since they do not require the shell to spawn a new process.

- **Example Command**: pwd

  $ type -t pwd

  Output: builtin



Command	Description
`pwd`	Print the current working directory.
`cd`	Change the current directory.
`echo`	Display a line of text/string.

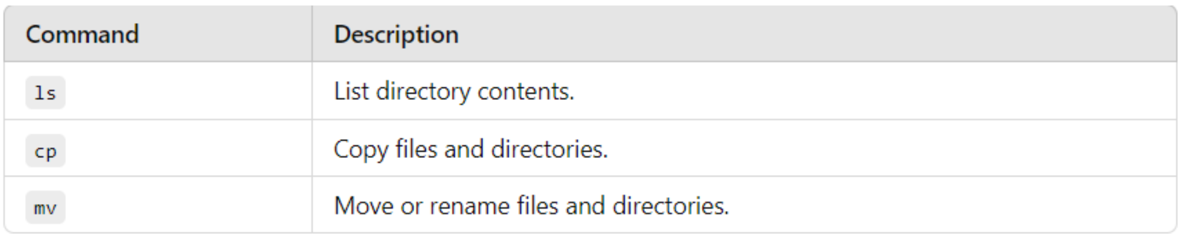
2.File Commands

File commands refer to external executable files located in the directories specified in the PATH environment variable. When you run an external command, the shell searches for the command in these directories.

- **Example Command**: ls

  $ type -t ls

  Output: file



Command	Description
`ls`	List directory contents.
`cp`	Copy files and directories.
`mv`	Move or rename files and directories.

3.Alias Commands

Aliases are user-defined shortcuts for commands, allowing for simplified command usage or customized behavior. They are defined in the shell environment to replace longer command syntax.

- **Example Command**: alias p='pwd'

  $ type -t p

  Output: alias

Command	Description
`alias p='pwd'`	Create an alias `p` for the `pwd` command.
`alias ll='ls -l'`	Create an alias `ll` for the `ls -l` command.
`unalias p`	Remove the alias `p`.

Which Command

The which command in Linux is used to identify the location of executables in your system's PATH. It helps you find out which directory a particular command resides in.

Example Explanation

- Command: which date
- Output: /bin/date
- This means that the date command is located in the /bin directory.

Action	Command	Example
Find the location of a command	`which <command>`	`which date`
Check multiple commands at once	`which <command1> <command2>`	`which ls pwd`
Get the location of a script	`which <script_name>`	`which my_script.sh`

Environment Variables

Environment variables are dynamic values that affect the behavior of processes in the operating system. They are used to store information about the environment in which a process runs, such as system paths, user preferences, and settings for applications.

Common Uses of Environment Variables

- Configure user preferences for shell sessions.
- Define paths for executable files and libraries.
- Store system information (e.g., the current user, home directory).
- Manage application settings.

Common Environment Variables

- **HOME**: The current user's home directory.
- **PATH**: A list of directories to search for executable files.

- **USER**: The name of the currently logged-in user.
- **SHELL**: The path to the current user's shell.
- **LANG**: The current language setting for the environment.

Action	Command	Example
Display all environment variables	`printenv` or `env`	`printenv` or `env`
Display a specific environment variable	`echo $VARIABLE`	`echo $HOME`
Set a new environment variable	`export VARIABLE=value`	`export MY_VAR="Hello"`
Modify an existing environment variable	`export VARIABLE=new_value`	`export PATH="$PATH:/new/directory"`
Unset an environment variable	`unset VARIABLE`	`unset MY_VAR`
Make a variable available to subprocesses	`export VARIABLE=value`	`export MY_VAR="Hello"` (available in subshells)
Check if a variable is set	`if [ -z "$VARIABLE" ]; then ...`	`if [ -z "$MY_VAR" ]; then echo "Not set"; fi`

Add Directory to path variable

Adding a directory to the PATH variable will enable you to call on your program or script from anywhere in the system, without needing to specify the path to where you've stored it.

1.Temporarily add a directory to $PATH

- To add a directory to $PATH for the current session ,use the following command syntax .In this example ,we 're adding the /bin/myscripts directory .
  $export PATH="/bin/myscripts:$PATH"

2. Permanently add a directory to $PATH

- To add a directory to $PATH permanently, we'll need to edit the **.bashrc** file of the user you want to change. Use nano or your favorite text editor to open the file, stored in the home directory.

  $ vi ~/.bashrc

- At the end of this file, put your new directory that you wish to permanently add to $PATH.

  $export PATH="/bin/myscripts:$PATH"

- Save your changes and exit the file. Afterwards, execute the following command to make the changes take effect in your current session. Alternative, you can log out or reboot the system.

```
$ source ~/.bashrc
```

- That's all there is to it. You can check $PATH once more to verify the change.

```
$ echo $PATH
```

Command Help

In Linux, man (manual) and help commands are used to access documentation and assistance for commands, utilities, and configuration options.

Help: The help command provides a brief summary of built-in shell commands and their options. It's mainly used for shell built-ins, not external commands

Action	Command	Example
Display help for a command	`<command> --help`	`ls --help`
List built-in shell commands	`help`	`help` (to see built-in command summary)
Get help for a specific built-in	`help <builtin_command>`	`help cd`

Command man

Man : The man command displays the manual pages for various commands and programs. It provides detailed information about the command, including its options, usage, and examples.

Action	Command	Example
Display the manual for a command	`man <command>`	`man ls`
View a specific section	`man <section> <command>`	`man 5 passwd` (view section 5 for the `passwd` command)
Search within man pages	`/search_term` (while in man)	`/copy` (search for "copy")
Go to the next search result	`n` (while in man)	`n` (to find the next occurrence of the search term)
Exit the man page	`q`	`q` (to quit the man viewer)

Redirects

In Linux, redirection is a way to control the input and output of commands. It allows you to send output to a file or receive input from a file instead of the terminal. Here's a breakdown of the most common types of redirection with examples:

1.Output Redirection (1)

Redirects the output of a command to a file instead of displaying it on the terminal.

Action	Command	Example
Redirect output to a file	`command > filename`	`echo "Hello, World!" > output.txt`
Append output to a file	`command >> filename`	`echo "Another line." >> output.txt`
Redirect output to /dev/null	`command > /dev/null`	`ls > /dev/null`

2.Input Redirection (0)

Takes input for a command from a file rather than the keyboard.

Action	Command	Example
Redirect input from a file	`command < filename`	`sort < unsorted.txt`

3.Error Redirection (2)

Redirects error messages from a command to a file or another command.

Action	Command	Example
Redirect error output to a file	`command 2> filename`	`ls non_existing_file 2> error.log`
Append error output to a file	`command 2>> filename`	`ls non_existing_file 2>> error.log`
Redirect both output and errors	`command > output.txt 2> error.txt`	`ls > output.txt 2> error.txt`
Redirect both output and errors to the same file	`command &> filename`	`ls &> all_output.txt`

4.Error Redirecting both error and output

You can redirect boh standard output and standard error to different files or the same file.

Redirection Type	Command Syntax	Example
Redirect Standard Error	`command 2> filename`	`ls non_existing_file 2> error.log`
Append Standard Error	`command 2>> filename`	`ls non_existing_file 2>> error.log`
Redirect Both Outputs (stderr to stdout)	`command > output.txt 2>&1`	`ls > all_output.txt 2>&1`
Append Both Outputs (stderr to stdout)	`command >> output.txt 2>&1`	`ls >> all_output.txt 2>&1`
Redirect Error to Discard	`command 2> /dev/null`	`ls non_existing_file 2> /dev/null`

Different types of users in linux

On a standalone Linux, there are five types of user accounts: root, superuser, sudo, regular, and service. This tutorial compares these types and lists their differences.

1.root account

The root account is the default highest privileged account. It is available on all Linux distros. Most

Linux distros keep this account locked or recommend not to use it for regular tasks.

2.Superuser account

Linux distros that do not lock the root account by default suggest using this account for system administration. They have a default privileged group called **wheel**. We can convert any regular user account into a super user account by adding that account to this group.

3.sudo user account

Linux distros that keep the root account locked use two commands to authenticate privileged tasks. These commands are the **sudo** and **su**.

The **sudo** command allows the user to run one privileged command. The **su** command allows the user to run privileged commands until the user uses the **exit** command to exit the privileged session.

4.Regular user account

Regular user accounts have moderate privileges. These accounts are available on all distros. You can create, manage, and delete them as per requirements.

5.Service account

Service accounts have the least privileges. Services use these accounts to run various processes. We cannot use these accounts to log in as a standard user. The installation process automatically creates necessary service accounts for default services. When we install a new software package or remove an existing one, the package installer automatically adds or removes related service accounts.

File Permissions

In Linux, file permissions determine who can read, write, or execute a file. Permissions are set for three types of users:

1. **Owner**: The user who owns the file.
2. **Group**: The group that owns the file.
3. **Others**: All other users.

Types of Permissions

- **Read (r)**: Allows viewing the contents of a file.
- **Write (w)**: Allows modifying or deleting the file.
- **Execute (x)**: Allows executing the file as a program or script.

Numeric Permission Values

- 0: --- (no permissions)
- 1: --x (execute only)
- 2: -w- (write only)
- 3: -wx (write and execute)
- 4: r-- (read only)
- 5: r-x (read and execute)
- 6: rw- (read and write)
- 7: rwx (read, write, and execute)

Understanding File Permission Notation

- Permissions are represented in a 10-character string:
  - The first character indicates the type (e.g., - for a regular file, d for a directory).
  - The next three characters represent the owner's permissions.
  - The next three characters represent the group's permissions.
  - The last three characters represent others' permissions.

Example: -rwxr-xr--

- -: Regular file
- rwx: Owner has read, write, and execute permissions
- r-x: Group has read and execute permissions
- r--: Others have read permissions only

Action	Command	Example	Description
View file permissions	`ls -l <file>`	`ls -l myfile.txt`	Displays detailed information, including permissions.
Change file permissions	`chmod <permissions> <file>`	`chmod u+x myscript.sh`	Adds execute permission for the owner.
Change ownership	`chown <user>:<group> <file>`	`chown user1:group1 myfile.txt`	Changes the owner and group of the file.
Change group ownership	`chgrp <group> <file>`	`chgrp group1 myfile.txt`	Changes the group ownership of the file.
Set permissions numerically	`chmod <numeric_permissions> <file>`	`chmod 755 myfile.txt`	Sets permissions using numeric notation (Owner: rwx, Group: r-x, Others: r--).

Copying files and Renaming

Operation	Command	Description
Copy a file	`cp source_file destination_file`	Copies `source_file` to `destination_file`.
Copy a directory	`cp -r source_directory destination_directory`	Copies `source_directory` recursively.
Rename a file	`mv old_filename new_filename`	Renames `old_filename` to `new_filename`.
Rename a directory	`mv old_directory_name new_directory_name`	Renames `old_directory_name` to `new_directory_name`.

Archiving and Compressing

Archiving: Archiving is the process of collecting multiple files and directories into a single file (archive) for easier storage, transfer, or backup. Common archiving formats include .tar and .zip.

Compressing: Compressing reduces the size of files or directories to save storage space or speed up transfer. Common compression tools include gzip, bzip2, and zip.

Action	Command	Example
Create a tar archive	`tar -cvf <archive_name>.tar <directory>`	`tar -cvf backup.tar /home/user/documents`
Extract a tar archive	`tar -xvf <archive_name>.tar`	`tar -xvf backup.tar`
Create a gzipped tar file	`tar -czvf <archive_name>.tar.gz <directory>`	`tar -czvf backup.tar.gz /home/user/documents`
Extract a gzipped tar file	`tar -xzvf <archive_name>.tar.gz`	`tar -xzvf backup.tar.gz`
Compress a file	`gzip <file>`	`gzip file.txt`
Decompress a file	`gunzip <file>.gz`	`gunzip file.txt.gz`
Create a bzip2 archive	`tar -cjvf <archive_name>.tar.bz2 <directory>`	`tar -cjvf backup.tar.bz2 /home/user/documents`
Extract a bzip2 archive	`tar -xjvf <archive_name>.tar.bz2`	`tar -xjvf backup.tar.bz2`
Create a zip archive	`zip <archive_name>.zip <files>`	`zip backup.zip file1.txt file2.txt`
Extract a zip archive	`unzip <archive_name>.zip`	`unzip backup.zip`

Soft Links / Hard Link

Hard Link: A hard link is a direct reference to the inode of a file. Multiple hard links to a file point to the same data on disk, and if the original file is deleted, the data remains accessible through the hard link.

Soft Link (Symbolic Link): A soft link is a pointer to another file or directory. It acts as a shortcut and can link to files on different filesystems. If the original file is deleted, the soft link becomes broken.

Operation	Command	Description
Create a Hard Link	`ln source_file hard_link_name`	Creates a hard link named `hard_link_name` pointing to `source_file`.
Create a Soft Link	`ln -s source_file soft_link_name`	Creates a symbolic link named `soft_link_name` pointing to `source_file`.
List Links	`ls -l`	Displays links; hard links have the same inode number, soft links show the path.
Remove a Link	`rm link_name`	Deletes the link (does not delete the original file for hard links if other links exist).

Text Processing Commands

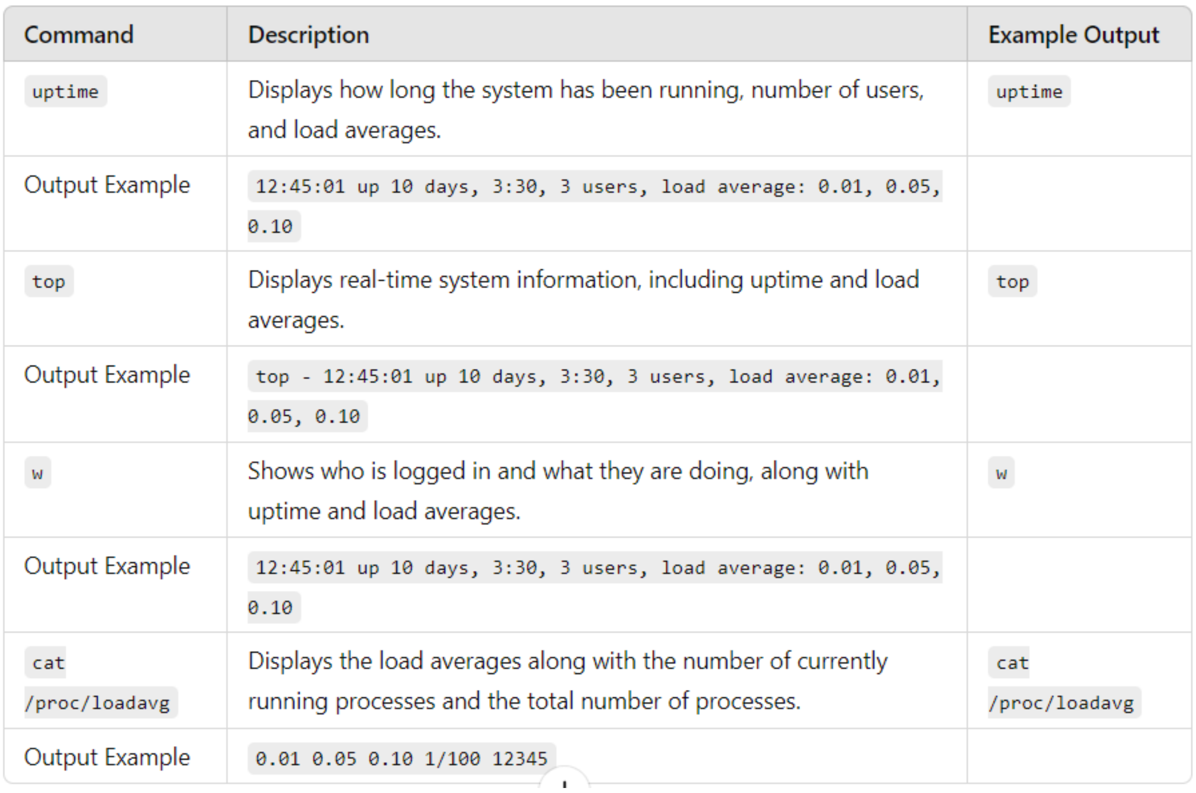
Command	Description	Example Command
`stdout`	Standard output; the default destination for output from command-line programs.	`echo "Hello, World!"` outputs to stdout.
`stdin`	Standard input; the default source of input for command-line programs, usually the keyboard or input redirection.	`cat < file.txt` reads from stdin (file.txt).
`stderr`	Standard error; the default destination for error messages from command-line programs.	`ls non_existing_file 2> error.log` redirects stderr.
`cut`	Used to remove sections from each line of files.	`cut -d':' -f1 /etc/passwd` extracts usernames.
`paste`	Merges lines of files.	`paste file1.txt file2.txt` combines lines side by side.
`head`	Outputs the first few lines of a file.	`head -n 10 file.txt` shows the first 10 lines.
`tail`	Outputs the last few lines of a file.	`tail -n 10 file.txt` shows the last 10 lines.
`expand`	Converts tabs to spaces in a file.	`expand file.txt` replaces tabs with spaces.
`unexpand`	Converts spaces to tabs in a file.	`unexpand file.txt` replaces spaces with tabs.
`join`	Joins lines of two files on a common field.	`join file1.txt file2.txt` combines lines based on a key.
`split`	Splits a file into multiple files.	`split -l 1000 largefile.txt` splits into 1000 lines each.
`sort`	Sorts lines of text files.	`sort file.txt` sorts lines alphabetically.
`tr`	Translates or deletes characters.	`echo "hello"
`uniq`	Removes duplicate lines from a sorted file.	`uniq file.txt` filters out adjacent duplicate lines.
`wc`	Counts lines, words, and characters in a file.	`wc file.txt` shows counts of lines, words, characters.

Server Review

1.Uptime and Load

Uptime: It indicates how long the system has been running since the last boot. It shows the current time, how long the system has been up, number of users logged in, and the system load averages.

Load average :It represents the average system load over a specified period (usually 1, 5, and 15 minutes). It indicates how many processes are actively competing for CPU time. A higher load average relative to the number of CPU cores may indicate the system is under heavy load.



Command	Description	Example Output
`uptime`	Displays how long the system has been running, number of users, and load averages.	`uptime`
Output Example	`12:45:01 up 10 days, 3:30, 3 users, load average: 0.01, 0.05, 0.10`	
`top`	Displays real-time system information, including uptime and load averages.	`top`
Output Example	`top - 12:45:01 up 10 days, 3:30, 3 users, load average: 0.01, 0.05, 0.10`	
`w`	Shows who is logged in and what they are doing, along with uptime and load averages.	`w`
Output Example	`12:45:01 up 10 days, 3:30, 3 users, load average: 0.01, 0.05, 0.10`	
`cat /proc/loadavg`	Displays the load averages along with the number of currently running processes and the total number of processes.	`cat /proc/loadavg`
Output Example	`0.01 0.05 0.10 1/100 12345`	

Explanation of Output

- **Current Time**: The current time of day.
- **Up Time**: How long the system has been running.
- **Number of Users**: How many users are currently logged into the system.
- **Load Average**: Three numbers indicating the load average over the last 1, 5, and 15 minutes, respectively.

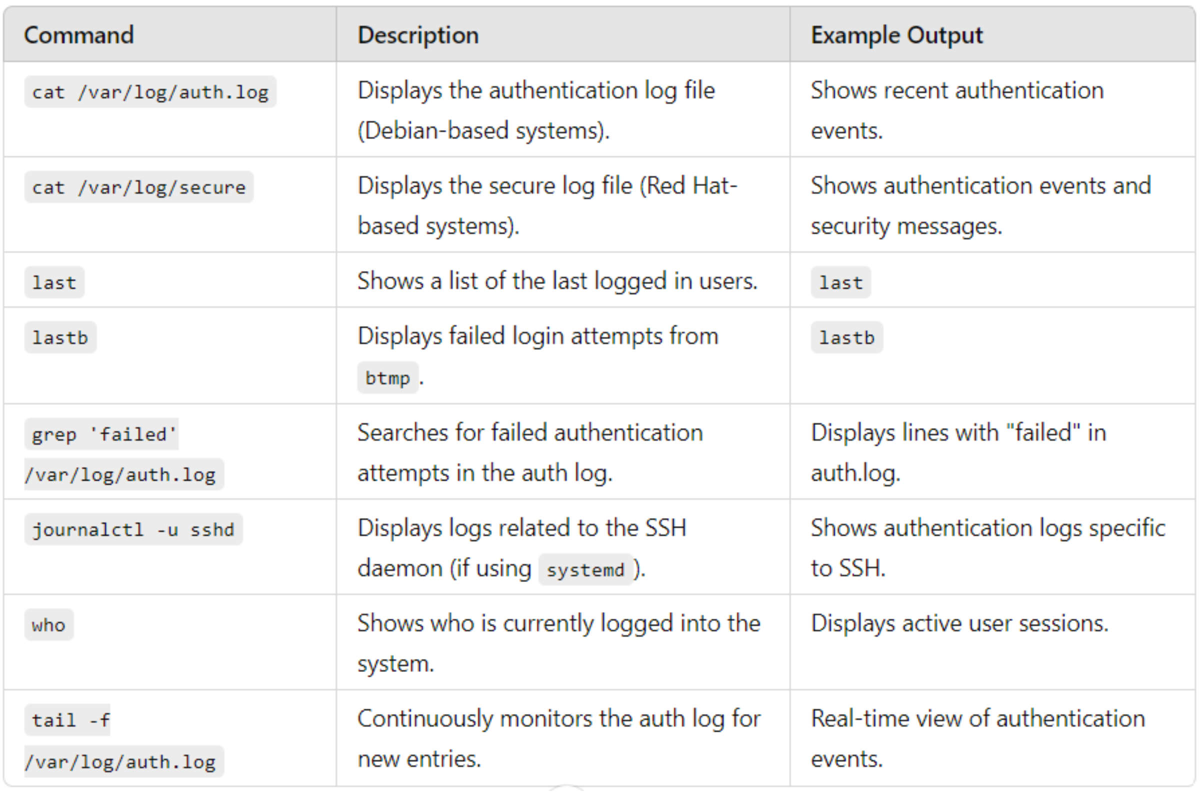
Load Average Interpretation

- **Load Average < Number of CPU Cores**: System is under normal load.
- **Load Average ≈ Number of CPU Cores**: System is under moderate load.
- **Load Average > Number of CPU Cores**: System may be overloaded, and performance could degrade.

2.Authentication Logs

Authentication logs in Linux track login attempts and other authentication-related events. These logs are crucial for security monitoring, as they help system administrators detect unauthorized access, failed login attempts, and other potentially malicious activities.

- **/var/log/auth.log**: Contains system authorization information, including user logins and authentication failures. Common on Debian-based systems (like Ubuntu).
- **/var/log/secure**: Similar to auth.log, this file stores authentication-related messages. Common on Red Hat-based systems (like CentOS and Fedora).
- **/var/log/lastlog**: Records the last login times of users.
- **/var/log/btmp**: Logs failed login attempts.



Command	Description	Example Output
`cat /var/log/auth.log`	Displays the authentication log file (Debian-based systems).	Shows recent authentication events.
`cat /var/log/secure`	Displays the secure log file (Red Hat-based systems).	Shows authentication events and security messages.
`last`	Shows a list of the last logged in users.	`last`
`lastb`	Displays failed login attempts from `btmp`.	`lastb`
`grep 'failed' /var/log/auth.log`	Searches for failed authentication attempts in the auth log.	Displays lines with "failed" in auth.log.
`journalctl -u sshd`	Displays logs related to the SSH daemon (if using `systemd`).	Shows authentication logs specific to SSH.
`who`	Shows who is currently logged into the system.	Displays active user sessions.
`tail -f /var/log/auth.log`	Continuously monitors the auth log for new entries.	Real-time view of authentication events.

Explanation of Common Log Entries

- **Accepted password for**: Indicates a successful login attempt.
- **Failed password for**: Indicates a failed login attempt.
- **session opened for user**: Indicates a session has been created for a user.
- **session closed for user**: Indicates a user session has ended.
- **Invalid user**: Shows attempts to log in with a non-existent username.

Importance of Authentication Logs

- **Security Monitoring**: Helps in detecting unauthorized access and potential breaches.

- **Auditing**: Provides a trail of user activities for compliance and auditing purposes.
- **Troubleshooting**: Assists in diagnosing login issues and understanding system access patterns.

Regularly monitoring these logs is an essential practice for maintaining the security and integrity of Linux systems.

3.Services Running

In Linux, "server running" typically refers to the operation of various server processes that handle requests and provide services to clients. These servers can include web servers, database servers, file servers, and more. Monitoring and managing these servers is crucial for ensuring availability and performance.

Key Concepts of Server Running

- **Service**: A background process that provides functionality, such as serving web pages or handling database queries.
- **Daemon**: A type of service that runs in the background, waiting for requests (e.g., httpd for web servers, sshd for SSH access).
- **Ports**: Services listen on specific network ports to accept incoming connections (e.g., HTTP on port 80, HTTPS on port 443).

Command	Description	Example Output
`systemctl status <service>`	Displays the status of a service managed by `systemd`.	`systemctl status apache2`
`systemctl start <service>`	Starts a specified service.	`systemctl start mysql`
`systemctl stop <service>`	Stops a specified service.	`systemctl stop mysql`
`systemctl restart <service>`	Restarts a specified service.	`systemctl restart nginx`
`systemctl enable <service>`	Enables a service to start at boot time.	`systemctl enable sshd`
`systemctl disable <service>`	Disables a service from starting at boot time.	`systemctl disable sshd`
`ps aux	grep <service>`	Lists all running processes and filters for the specified service.

`netstat -tuln`	Displays active listening ports and associated services.	Shows which services are listening on which ports.
`lsof -i :<port>`	Lists open files and processes using the specified port.	`lsof -i :80`
`curl -I http://localhost`	Checks if a web server is running by sending a HEAD request.	Response headers from the web server.
`journalctl -u <service>`	Displays logs related to a specific service.	`journalctl -u nginx`
`top`	Displays real-time resource usage, including running server processes.	Real-time view of CPU and memory usage.
`htop`	An improved, interactive version of `top` for monitoring system processes.	Real-time view of system processes with more details.

Importance of Monitoring Server Status

- **Availability**: Ensures that services are running and accessible to users.
- **Performance**: Helps identify resource bottlenecks and optimize server performance.
- **Troubleshooting**: Assists in diagnosing issues with server processes and connectivity.
- **Security**: Monitoring can help detect unauthorized service changes or access attempts.

Regularly managing and monitoring your servers is essential for maintaining the reliability and security of your services in a Linux environment

4.Available Memory / Disk

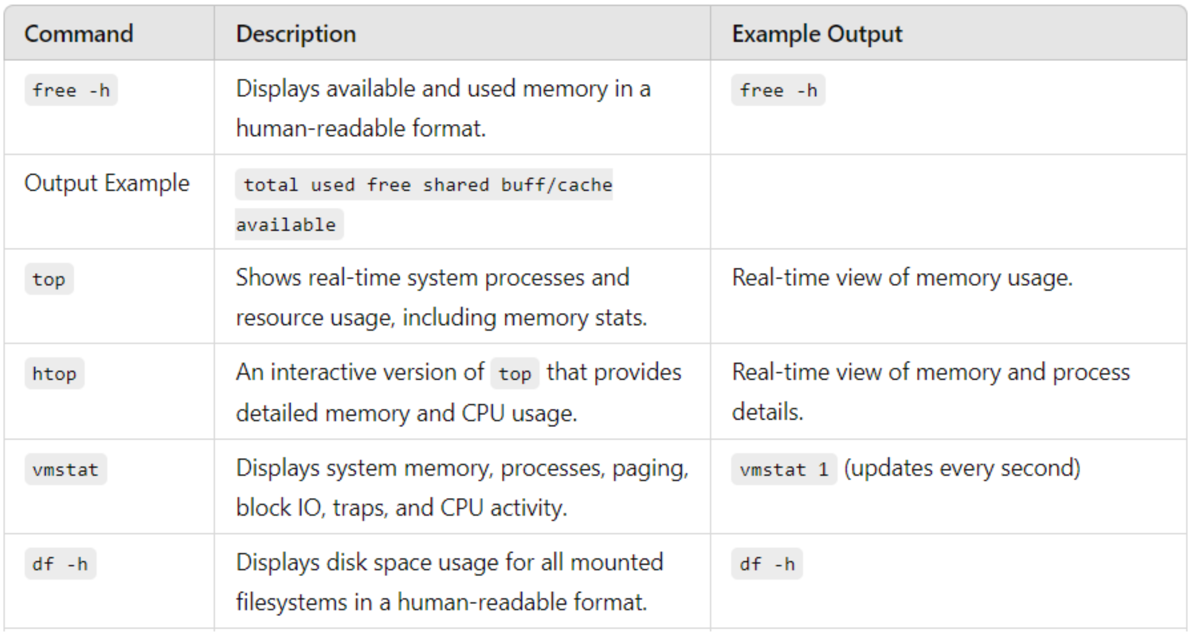
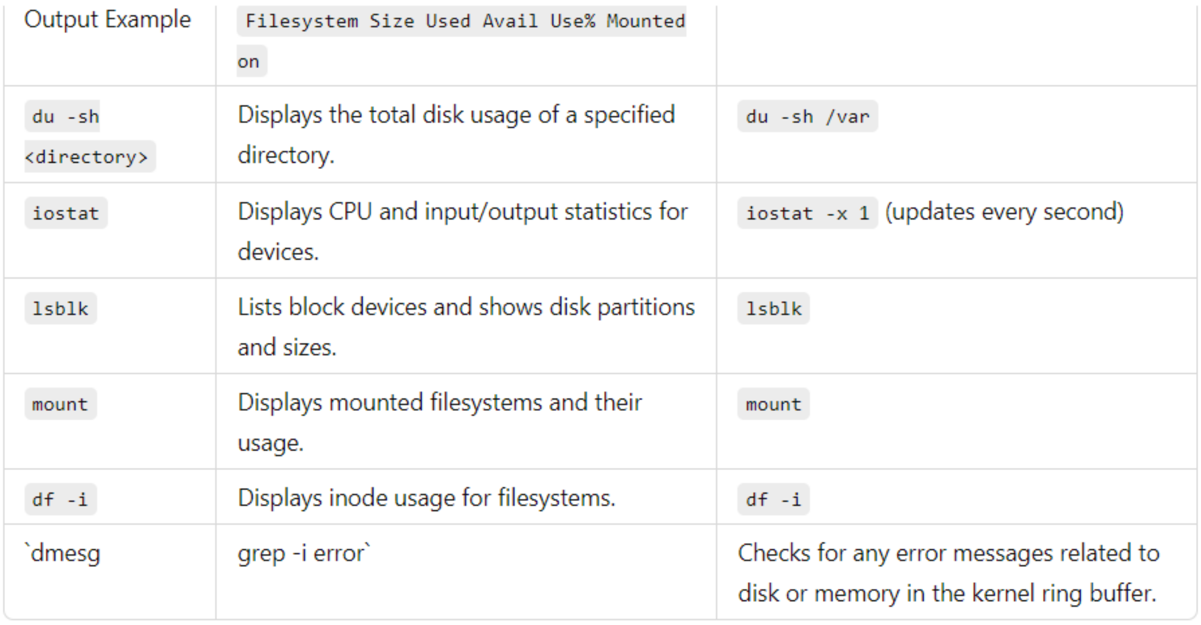
. Monitoring available memory and disk space in Linux is crucial for ensuring system performance and stability. Here’s an overview of available memory and disk usage, along with relevant commands.

Available Memory

Available memory refers to the amount of RAM that is not currently being used by running processes and can be allocated for new tasks. Insufficient memory can lead to system slowdowns and application failures.

Disk Space

Disk space refers to the total storage capacity of a system and how much of it is currently in use. Running out of disk space can lead to application failures, data loss, and system instability.

Command	Description	Example Output
`free -h`	Displays available and used memory in a human-readable format.	`free -h`
Output Example	`total used free shared buff/cache available`	
`top`	Shows real-time system processes and resource usage, including memory stats.	Real-time view of memory usage.
`htop`	An interactive version of `top` that provides detailed memory and CPU usage.	Real-time view of memory and process details.
`vmstat`	Displays system memory, processes, paging, block IO, traps, and CPU activity.	`vmstat 1` (updates every second)
`df -h`	Displays disk space usage for all mounted filesystems in a human-readable format.	`df -h`
Output Example	`Filesystem Size Used Avail Use% Mounted on`	
`du -sh <directory>`	Displays the total disk usage of a specified directory.	`du -sh /var`
`iostat`	Displays CPU and input/output statistics for devices.	`iostat -x 1` (updates every second)
`lsblk`	Lists block devices and shows disk partitions and sizes.	`lsblk`
`mount`	Displays mounted filesystems and their usage.	`mount`
`df -i`	Displays inode usage for filesystems.	`df -i`
`dmesg	grep -i error`	Checks for any error messages related to disk or memory in the kernel ring buffer.

Importance of Monitoring Memory and Disk Usage

- **Performance**: Ensures that sufficient resources are available for applications to run efficiently.
- **Stability**: Helps prevent crashes and slowdowns caused by low memory or disk space.
- **Capacity Planning**: Assists in predicting when upgrades or expansions may be needed.
- **Security**: Monitoring can help detect unusual spikes in resource usage, potentially indicating security issues.

Process Management

Background / Foreground Processes

Foreground processes run in the active terminal session and can interact with it.

Background processes run behind the scenes, allowing you to continue using the terminal.

Action	Command	Example
Start a process in the background	`command &`	`sleep 60 &`
Bring a background process to the foreground	`fg %job_number`	`fg %1` (bring job 1 to foreground)
List current jobs	`jobs`	`jobs`

Listing / Finding Processes

You can list all running processes and find specific ones using various commands.

Action	Command	Example
List all processes	`ps aux`	`ps aux`
List processes in tree format	`ps -ejH`	`ps -ejH`
Find a specific process	`pgrep process_name`	`pgrep firefox`
Detailed process info	`top` or `htop`	`top` or `htop`

Process Signals

Signals are used to communicate with processes, such as asking them to terminate.

Action	Command	Example
Send a signal to a process	`kill -signal pid`	`kill -9 1234` (force kill PID 1234)
List available signals	`kill -l`	`kill -l`

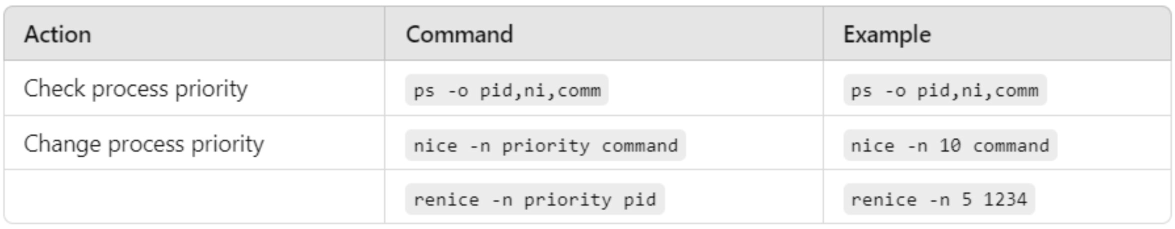
Killing Processes

You can terminate processes using their PID (Process ID).

Action	Command	Example
Kill a process by PID	`kill pid`	`kill 1234`
Force kill a process	`kill -9 pid`	`kill -9 1234`

Process Priorities

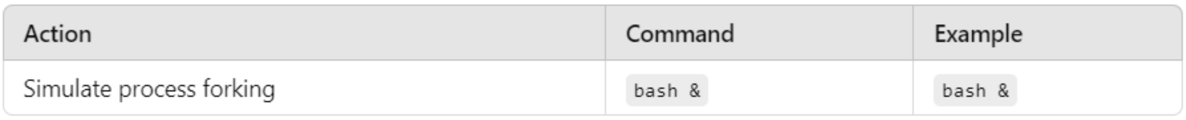
Each process has a priority that determines how much CPU time it receives.



Action	Command	Example
Check process priority	`ps -o pid,ni,comm`	`ps -o pid,ni,comm`
Change process priority	`nice -n priority command`	`nice -n 10 command`
	`renice -n priority pid`	`renice -n 5 1234`

Process Forking

Forking is the creation of a new process by an existing one, allowing for multitasking.



Action	Command	Example
Simulate process forking	`bash &`	`bash &`

User Management

Create / Delete / Update

Managing users involves adding, removing, or updating their information (like passwords or shell settings) on the system.

Users and Groups

A group allows managing permissions for multiple users. Each user can belong to one or more groups.

Managing Permissions

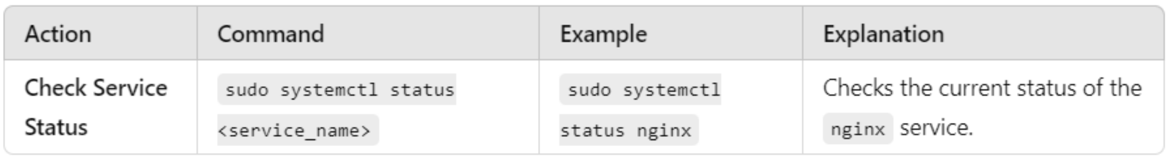
Permissions define what a user or group can do with a file or directory (read, write, execute). This is managed using ownership (user/group) and permission flags.

Action	Command	Example	Explanation
Create a User	`sudo useradd <username>`	`sudo useradd john`	Adds a new user with the username "john".
Delete a User	`sudo userdel <username>`	`sudo userdel john`	Deletes the user "john" from the system.
Modify a User	`sudo usermod <option> <username>`	`sudo usermod -s /bin/bash john`	Modifies "john" to use `/bin/bash` as their shell.
Set User Password	`sudo passwd <username>`	`sudo passwd john`	Sets or updates the password for "john".
Create a Group	`sudo groupadd <groupname>`	`sudo groupadd developers`	Creates a new group called "developers".
Delete a Group	`sudo groupdel <groupname>`	`sudo groupdel developers`	Deletes the group "developers".
Add User to Group	`sudo usermod -aG <groupname> <username>`	`sudo usermod -aG developers john`	Adds "john" to the group "developers".
View User Groups	`groups <username>`	`groups john`	Lists all groups "john" is a part of, including their primary group.
Change File Ownership	`sudo chown <user>:<group> <file>`	`sudo chown john:developers /var/www/html/index.html`	Changes the ownership of `index.html` to user "john" and group "developers".
Change File Permissions	`chmod <permissions> <file>`	`chmod 755 /var/www/html/index.html`	Sets permissions for `index.html` to rwxr-xr-x (owner has full access, group and others can only read/execute).
View File Permissions	`ls -l <file>`	`ls -l /var/www/html/index.html`	Displays the permissions and ownership of `index.html`.
Change Group Ownership	`sudo chgrp <group> <file>`	`sudo chgrp developers /var/www/html/index.html`	Changes the group ownership of `index.html` to "developers".
Change User Default Shell	`sudo usermod -s /bin/bash <username>`	`sudo usermod -s /bin/bash john`	Changes the default shell for "john" to bash.

Service Management (systemd)

Checking Service Status

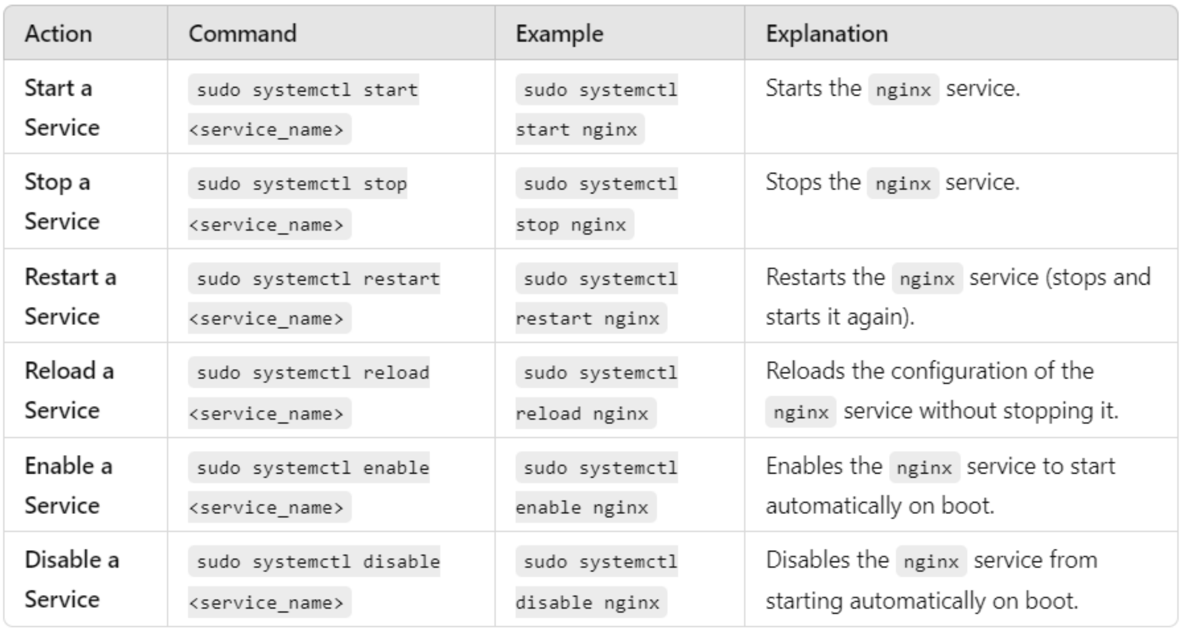
View whether a service is running, stopped, or in an error state.



Action	Command	Example	Explanation
Check Service Status	`sudo systemctl status <service_name>`	`sudo systemctl status nginx`	Checks the current status of the `nginx` service.

Start / Stop Services

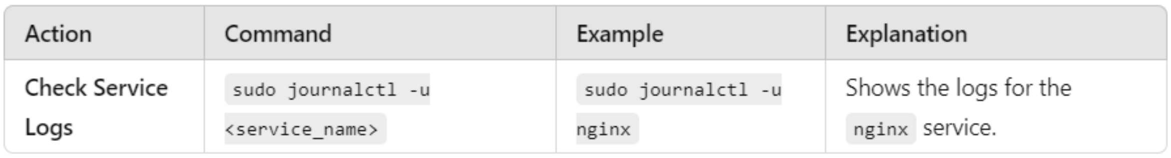
Manually start or stop services as needed.



Action	Command	Example	Explanation
Start a Service	`sudo systemctl start <service_name>`	`sudo systemctl start nginx`	Starts the `nginx` service.
Stop a Service	`sudo systemctl stop <service_name>`	`sudo systemctl stop nginx`	Stops the `nginx` service.
Restart a Service	`sudo systemctl restart <service_name>`	`sudo systemctl restart nginx`	Restarts the `nginx` service (stops and starts it again).
Reload a Service	`sudo systemctl reload <service_name>`	`sudo systemctl reload nginx`	Reloads the configuration of the `nginx` service without stopping it.
Enable a Service	`sudo systemctl enable <service_name>`	`sudo systemctl enable nginx`	Enables the `nginx` service to start automatically on boot.
Disable a Service	`sudo systemctl disable <service_name>`	`sudo systemctl disable nginx`	Disables the `nginx` service from starting automatically on boot.

Checking Service Logs

View logs related to a specific service to diagnose issues.



Action	Command	Example	Explanation
Check Service Logs	`sudo journalctl -u <service_name>`	`sudo journalctl -u nginx`	Shows the logs for the `nginx` service.

Creating New Services

Define custom services with a systemd service file.

Action	Command	Example	Explanation
Create a New Service	`sudo nano /etc/systemd/system/<service_name>.service`	`sudo nano /etc/systemd/system/myapp.service`	Opens a text editor to create a custom service file for `myapp`.
Reload systemd	`sudo systemctl daemon-reload`	`sudo systemctl daemon-reload`	Reloads systemd to apply changes after modifying or adding service files.
Start New Service	`sudo systemctl start <service_name>`	`sudo systemctl start myapp`	Starts the newly created service (`myapp`).

Package Management

Package Repositories

These are storage locations from where the Linux package manager retrieves software packages. Each distribution (like Ubuntu, CentOS) uses different repositories.

Command	Example
`sudo add-apt-repository <repository>`	`sudo add-apt-repository ppa:graphics-drivers/ppa`
`sudo apt update`	`sudo apt update`
`sudo apt-cache policy` (Debian/Ubuntu)	`sudo apt-cache policy`
`yum repolist` (CentOS)	`yum repolist`
`sudo add-apt-repository -r <repository>`	`sudo add-apt-repository -r ppa:graphics-drivers/ppa`

Snap

A universal Linux packaging format that works across all distributions, allowing easy installation and sandboxing of applications.

Command	Example
`sudo apt install snapd`	`sudo apt install snapd`
`sudo snap install <package_name>`	`sudo snap install vlc`
`snap list`	`snap list`
`sudo snap remove <package_name>`	`sudo snap remove vlc`

Finding and Installing Packages

Searching for specific software and installing it on your system.

Command	Example
`apt search <package_name>` (Debian/Ubuntu)	`apt search git`
`yum search <package_name>` (CentOS)	`yum search git`
`sudo apt install <package_name>` (Debian/Ubuntu)	`sudo apt install git`
`sudo yum install <package_name>` (CentOS)	`sudo yum install git`

Listing Installed Packages

Viewing all the packages that are currently installed on your system.

Command	Example
`dpkg --list` (Debian/Ubuntu)	`dpkg --list`
`rpm -qa` (CentOS)	`rpm -qa`
`dpkg -s <package_name>` (Debian/Ubuntu)	`dpkg -s git`
`rpm -q <package_name>` (CentOS)	`rpm -q git`

Install / Remove / Upgrade Packages

Commands to manage software installation, removal, and upgrading of packages.

Command	Example
`sudo apt install <package_name>` (Debian/Ubuntu)	`sudo apt install curl`
`sudo yum install <package_name>` (CentOS)	`sudo yum install curl`
`sudo apt remove <package_name>` (Debian/Ubuntu)	`sudo apt remove curl`
`sudo yum remove <package_name>` (CentOS)	`sudo yum remove curl`
`sudo apt upgrade` (Debian/Ubuntu)	`sudo apt upgrade`
`sudo yum update` (CentOS)	`sudo yum update`
`sudo apt autoremove`	`sudo apt autoremove`

Disks and Filesystems

Inodes

Inodes store metadata about files (such as file size, permissions, and pointers to data blocks) but not the file name or contents.

Command	Example
Display inode usage	`df -i`
Show inode info of a file	`ls -i <file_name>`
Check inode information on a filesystem	`stat <file_name>`

Filesystems

Filesystems determine how data is stored and retrieved. Common filesystems include ext4, xfs, and ntfs.

Command	Example
Check filesystem type	`df -T`
Create a filesystem	`mkfs -t <fs_type> <device>`
Check filesystem details	`sudo blkid`

Mounts

Mounting is the process of attaching a filesystem to a specific directory in the Linux file hierarchy.

Command	Example
List mounted filesystems	`mount`
Mount a filesystem	`sudo mount <device> <mount_point>`
Unmount a filesystem	`sudo umount <mount_point>`
Mount all filesystems from `/etc/fstab`	`sudo mount -a`

Adding Disks

Adding disks involves attaching a new disk and preparing it for use by partitioning, formatting, and mounting it.

Command	Example
List available disks	`lsblk`
Partition a disk	`sudo fdisk <device>`
Format the partition	`sudo mkfs -t <fs_type> <partition>`
Add a disk to `/etc/fstab` for auto-mounting	Edit `/etc/fstab`

Swap

Swap is a space on a disk used when the amount of physical RAM is full. It acts as virtual memory for the system.

Command	Example
Check swap space	`swapon --show`
Create a swap file	`sudo fallocate -l <size> /swapfile`
Set up swap file	`sudo mkswap /swapfile`
Enable swap	`sudo swapon /swapfile`
Disable swap	`sudo swapoff /swapfile`

LVM

LVM provides a more flexible way to manage disk storage by allowing dynamic resizing of partitions, snapshots, and volume groups.

Command	Example
Display LVM physical volumes (PVs)	`sudo pvdisplay`
Create a physical volume	`sudo pvcreate <device>`
Create a volume group (VG)	`sudo vgcreate <vg_name> <pv_name>`
Create a logical volume (LV)	`sudo lvcreate -L <size> -n <lv_name> <vg_name>`
Extend a logical volume	`sudo lvextend -L +<size> <lv_name>`

Booting Linux

The Linux boot process consists of several stages: BIOS/UEFI initialization, loading of the bootloader (like GRUB), kernel loading, and starting system services via systemd or init.

Command	Example
Check boot messages	`dmesg`
Display boot time and uptime	`uptime`
View system boot log	`journalctl -b`
List previous boot logs	`journalctl --list-boots`

Logs

Linux logs contain system and application information. The logs are typically stored in /var/log and are used to monitor system activities and troubleshoot issues.

Command	Example
View system logs	`journalctl`
View kernel logs	`dmesg`
View a specific log file	`cat /var/log/<log_file>`
Monitor logs in real-time	`tail -f /var/log/<log_file>`
Show recent journal logs	`journalctl -xe`

Boot Loaders

Boot loaders like **GRUB** (Grand Unified Bootloader) are responsible for loading the Linux kernel and transferring control to the operating system during boot.

Command	Example
Update GRUB configuration	`sudo update-grub` (Debian/Ubuntu)
	`sudo grub2-mkconfig -o /boot/grub2/grub.cfg` (CentOS)
List GRUB boot entries	`grep menuentry /boot/grub/grub.cfg`
Install GRUB bootloader	`sudo grub-install <device>`
Edit GRUB configuration file	`sudo nano /etc/default/grub`
Set default boot entry	`sudo grub-set-default <entry_number>`

Networking

TCP/IP Stack

The TCP/IP stack is a set of protocols used for network communication. It includes four layers: Application, Transport, Internet, and Network Interface, allowing different devices to communicate over the internet.

Command	Example
View network interfaces	`ip addr`
Check TCP/IP configuration	`ifconfig` or `ip addr show`
Test network connectivity	`ping <hostname_or_IP>`

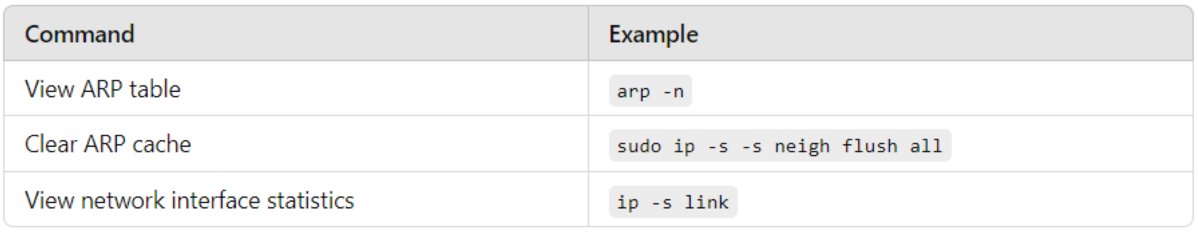
Subnetting

Subnetting divides an IP network into smaller, manageable sub-networks (subnets) to improve performance and security.

Command	Example
Calculate subnet mask	Use a subnet calculator tool or online resource
Display subnet information	`ip route`
Check subnet IP addresses	`ipcalc <ip_address>`

Ethernet & arp/rarp

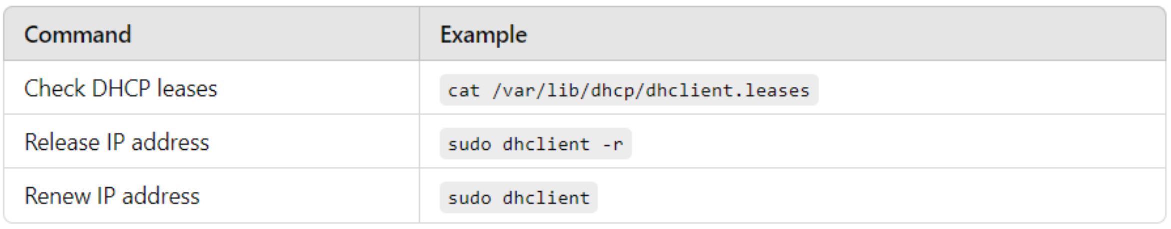
Ethernet is a common networking technology for local area networks (LANs). ARP (Address Resolution Protocol) maps IP addresses to MAC addresses, while RARP (Reverse ARP) does the opposite.



Command	Example
View ARP table	`arp -n`
Clear ARP cache	`sudo ip -s -s neigh flush all`
View network interface statistics	`ip -s link`

DHCP

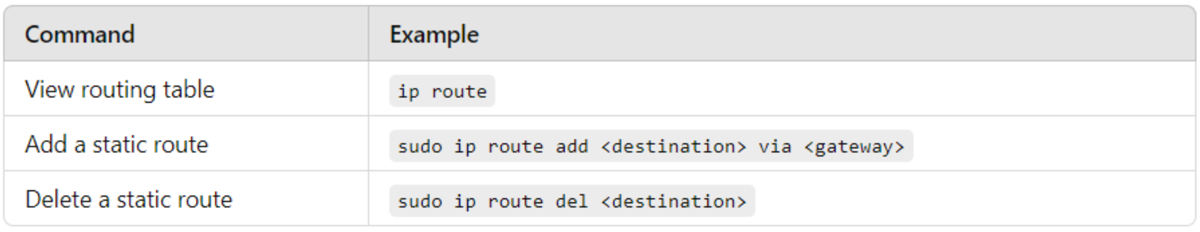
Dynamic Host Configuration Protocol (DHCP) automatically assigns IP addresses and other network configuration parameters to devices on a network.



Command	Example
Check DHCP leases	`cat /var/lib/dhcp/dhclient.leases`
Release IP address	`sudo dhclient -r`
Renew IP address	`sudo dhclient`

IP Routing

IP routing determines how data packets travel across networks from the source to the destination. It involves the use of routing tables.



Command	Example
View routing table	`ip route`
Add a static route	`sudo ip route add <destination> via <gateway>`
Delete a static route	`sudo ip route del <destination>`

DNS Resolution

Domain Name System (DNS) translates human-readable domain names (like www.example.com) into IP addresses, allowing browsers to load Internet resources.

Command	Example
Check DNS resolution	`nslookup <domain>`
View DNS information	`dig <domain>`
Check DNS cache	`sudo systemd-resolve --status`

Netfilter

Netfilter is a framework in Linux that provides firewall functionality through iptables. It allows for packet filtering, network address translation (NAT), and more.

Command	Example
View iptables rules	`sudo iptables -L`
Add a rule to allow traffic	`sudo iptables -A INPUT -p tcp --dport <port> -j ACCEPT`
Save iptables rules	`sudo iptables-save > /etc/iptables/rules.v4`

SSH

Secure Shell (SSH) is a protocol used to securely access remote machines over a network.

Command	Example
Connect to a remote server	`ssh <username>@<hostname_or_IP>`
Copy files using SSH	`scp <local_file> <username>@<hostname>:<remote_path>`
Generate SSH key pair	`ssh-keygen`

File transfer

Linux supports various file transfer methods, including SCP, SFTP, and FTP, allowing users to send and receive files over the network.

Command	Example
Transfer files using SCP	`scp <local_file> <username>@<hostname>:<remote_path>`
Connect using SFTP	`sftp <username>@<hostname>`
Transfer files using FTP	`ftp <hostname>`

Backup Tools

Backup tools in Linux are used to create copies of files and directories to prevent data loss. Common tools include tar, rsync, and cp.

Command	Example
Create a tar archive	`tar -cvf <archive_name>.tar <directory>`
Extract a tar archive	`tar -xvf <archive_name>.tar`
Backup files using rsync	`rsync -av <source> <destination>`

Shell Programming

Shell programming involves writing scripts using a shell (like Bash) to automate tasks. Shell scripts can include commands, control flow, and functions.

Command	Example
Create a new shell script	`nano script.sh`
Make the script executable	`chmod +x script.sh`
Run a shell script	`./script.sh`

Debugging

Debugging in shell scripts involves identifying and fixing errors. Tools like set -x help in tracing script execution.

Command	Example
Enable debugging mode	`set -x`
Disable debugging mode	`set +x`
Use `echo` to debug variables	`echo $variable`

Conditionals

Conditionals in shell scripting allow scripts to make decisions based on conditions using if, else, and case statements.

Command	Example
Basic if statement	`if [ condition ]; then commands; fi`
If-else statement	`if [ condition ]; then commands; else commands; fi`
Case statement	`case variable in pattern) commands ;; esac`

Loops

Loops allow scripts to execute commands repeatedly based on conditions. Common types are for, while, and until loops.

Command	Example
For loop	`for variable in list; do commands; done`
While loop	`while [ condition ]; do commands; done`
Until loop	`until [ condition ]; do commands; done`

Literals Variables

Literals are fixed values used directly in scripts (like numbers and strings). Variables store values for later use.

Command	Example
Declare a variable	`variable=value`
Access a variable	`echo $variable`
Use literal strings	`echo "This is a string"`

Troubleshooting

Troubleshooting network issues often involves a variety of tools that help diagnose and resolve connectivity problems. Here’s a brief overview of some key tools used in network troubleshooting.

ICMP

ICMP (Internet Control Message Protocol) is a network layer protocol used for error reporting and operational information exchange between network devices. It’s often used by diagnostic tools like ping and traceroute.

Ping

The ping command sends ICMP Echo Request messages to a specified host and waits for a response. It’s primarily used to check if a host is reachable over the network.

Command	Example	Description
Basic Ping	`ping <hostname or IP>`	Sends ICMP Echo Requests to the host.
Specify Number of Pings	`ping -c <count> <hostname>`	Limits the number of Echo Requests.
Continuous Ping	`ping -i <interval> <hostname>`	Sends packets at a specified interval.

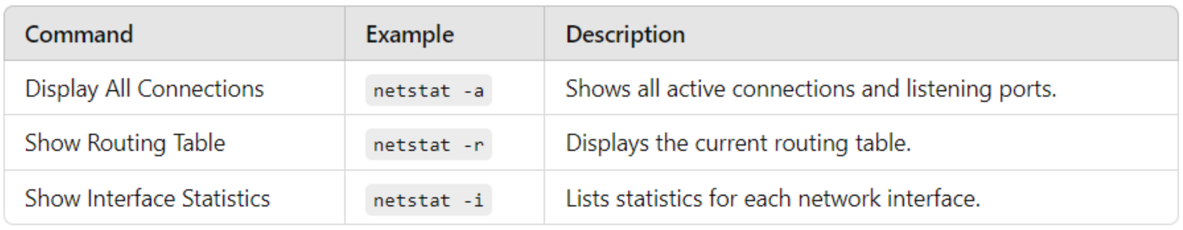
traceroute

The traceroute command tracks the path packets take to reach a destination, helping identify where delays or failures occur in the network.

Command	Example	Description
Basic Traceroute	`traceroute <hostname or IP>`	Displays the route taken by packets.
Use Specific Protocol	`traceroute -I <hostname>`	Uses ICMP Echo Requests instead of UDP.
Specify Maximum Hops	`traceroute -m <max_hops> <hostname>`	Limits the number of hops.

netstat

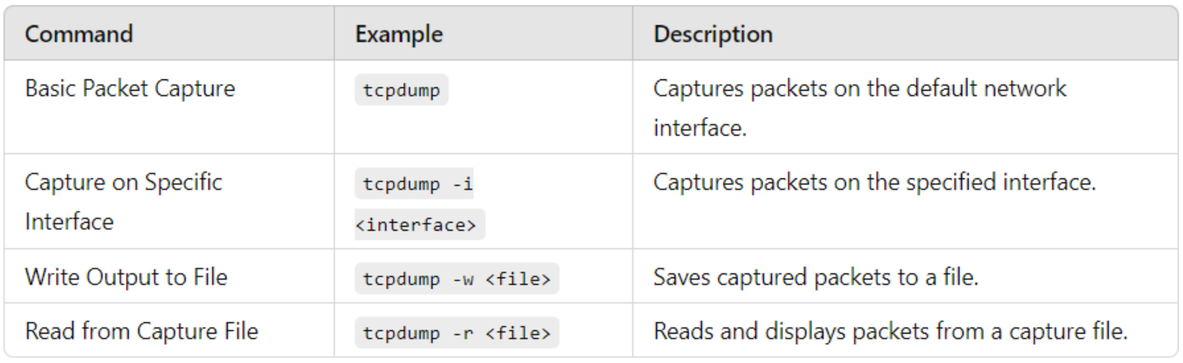
The netstat command provides network statistics, including current connections, routing tables, and interface statistics. It helps diagnose network issues and monitor traffic.



Command	Example	Description
Display All Connections	`netstat -a`	Shows all active connections and listening ports.
Show Routing Table	`netstat -r`	Displays the current routing table.
Show Interface Statistics	`netstat -i`	Lists statistics for each network interface.

Packet Analysis

Packet analysis involves capturing and analyzing data packets on the network. Tools like tcpdump and Wireshark are commonly used for this purpose.



Command	Example	Description
Basic Packet Capture	`tcpdump`	Captures packets on the default network interface.
Capture on Specific Interface	`tcpdump -i <interface>`	Captures packets on the specified interface.
Write Output to File	`tcpdump -w <file>`	Saves captured packets to a file.
Read from Capture File	`tcpdump -r <file>`	Reads and displays packets from a capture file.

Containerization

Containerization is a lightweight form of virtualization that allows you to run applications and their dependencies in isolated environments called containers. This technology helps ensure that applications run consistently across different computing environments. Here's a brief overview of some key concepts and tools in containerization.

ulimits

ulimits (user limits) are system settings that control the resources available to processes started by a shell. They help prevent any single process from consuming too many resources (like memory or CPU), which can lead to system instability.

Command	Example	Description
Show Current Limits	`ulimit -a`	Displays all current user limits.
Set Maximum Number of Open Files	`ulimit -n 1024`	Sets the maximum number of open files to 1024.
Set Maximum Process Size	`ulimit -v 500000`	Sets the maximum virtual memory size to 500000 KB.

Cgroups

cgroups is a Linux kernel feature that limits, accounts for, and isolates resource usage (CPU, memory, disk I/O, etc.) of a group of processes. It enables fine-grained resource management and is essential for running containers efficiently.

Command	Example	Description
Create a cgroup	`mkdir /sys/fs/cgroup/memory/mygroup`	Creates a new cgroup for memory management.
Limit Memory Usage	`echo 500M > /sys/fs/cgroup/memory/mygroup/memory.limit_in_bytes`	Limits memory usage to 500 MB.
Check Memory Usage	`cat /sys/fs/cgroup/memory/mygroup/memory.usage_in_bytes`	Displays current memory usage of the cgroup.

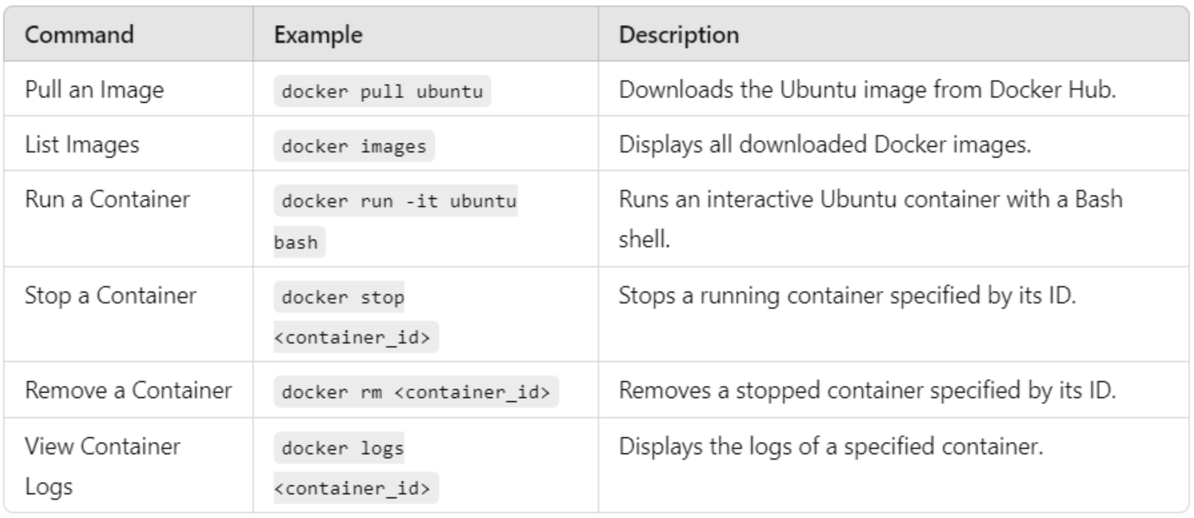
Container Runtime

A container runtime is software that is responsible for running containers. It provides the necessary environment for containers to run and manages their lifecycle. Examples include Docker, containerd, and CRI-O.

Command	Example	Description
Check Installed Runtimes	`crictl runtime`	Lists installed container runtimes (if applicable).
Run a Container	`docker run hello-world`	Runs a simple container using Docker.
Manage Container Lifecycle	`docker ps`	Lists running containers.

Docker

Docker is a popular platform for developing, shipping, and running applications in containers. It simplifies the process of managing containers by providing a command-line interface and a graphical user interface. Docker uses a client-server architecture to manage containers and images.



Command	Example	Description
Pull an Image	`docker pull ubuntu`	Downloads the Ubuntu image from Docker Hub.
List Images	`docker images`	Displays all downloaded Docker images.
Run a Container	`docker run -it ubuntu bash`	Runs an interactive Ubuntu container with a Bash shell.
Stop a Container	`docker stop <container_id>`	Stops a running container specified by its ID.
Remove a Container	`docker rm <container_id>`	Removes a stopped container specified by its ID.
View Container Logs	`docker logs <container_id>`	Displays the logs of a specified container.

Ulimits: Controls resource limits for processes.

Cgroups: Manages resource allocation for groups of processes.

Container Runtime: Software for running containers (e.g., Docker).

Docker: A platform for container management and application deployment.