# Introduction to Linux

Master the essential commands and concepts that power the world's most popular server operating system

CHAPTER 1

# Why Linux Matters

Linux powers over 90% of the world's servers, from Google and Facebook to your favorite streaming services. Understanding Linux isn't just useful—it's essential for anyone pursuing a career in IT, cloud computing, or systems administration.

Whether you're managing a single server or orchestrating thousands in the cloud, the commands and concepts you'll learn here form the foundation of modern infrastructure. Let's dive into the practical skills that will make you confident on the command line.

# What Makes Linux Special?

### Open Source Freedom

View, modify, and distribute the source code freely. Learn how your OS actually works under the hood.

### Rock-Solid Security

Built-in permission systems and regular security updates keep your systems protected from threats.

### Maximum Performance

Minimal resource usage means more power for your applications. Perfect for servers and embedded systems.

### Vibrant Community

Millions of developers worldwide contribute, support, and share knowledge to help you succeed.

CHAPTER 2

# Essential Linux Commands

Every Linux journey begins with mastering the command line. These fundamental commands are your tools for navigating, managing, and controlling your system. Think of them as the vocabulary of Linux—once you know them, you can accomplish anything.

# Navigation & File Basics

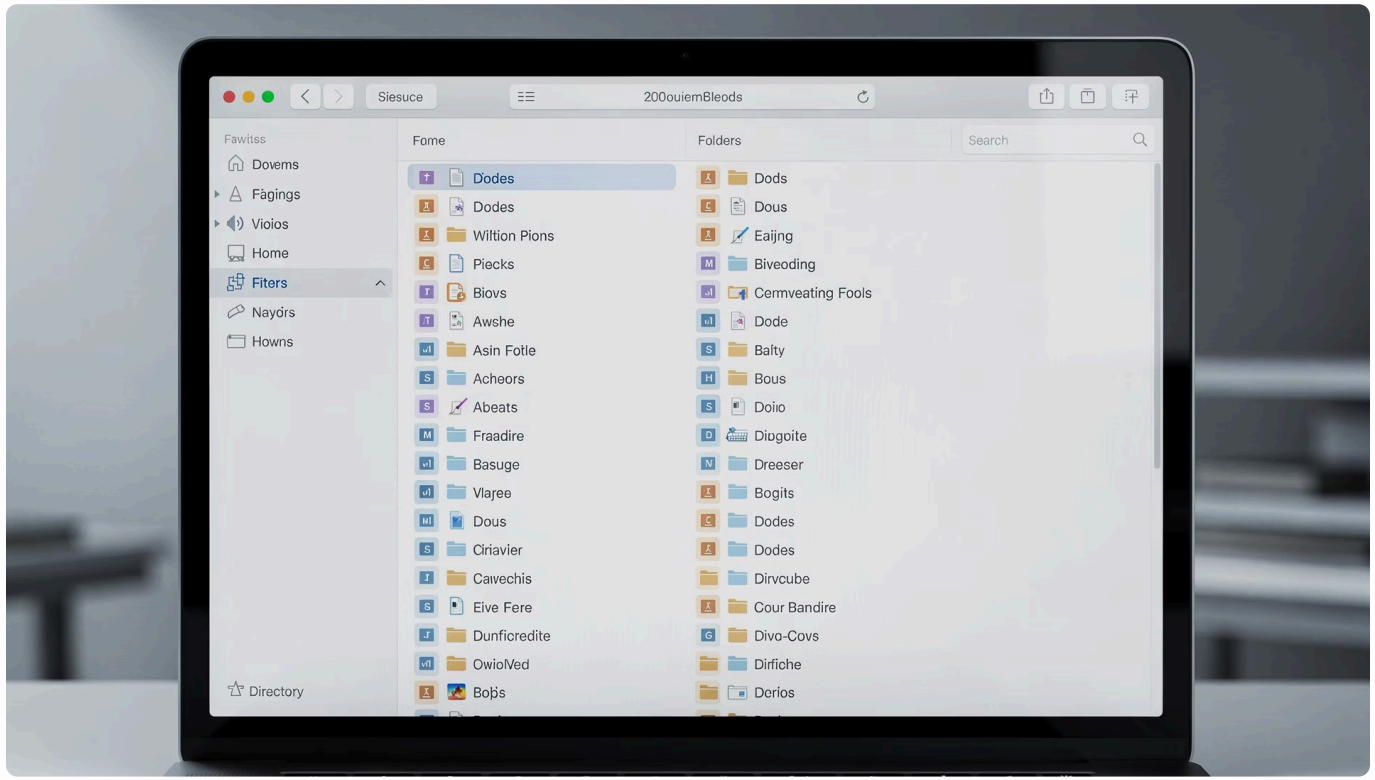
## Your First Commands

These commands help you move around the file system and understand where you are. Practice these until they become second nature.

**pwd** shows your current directory—think of it as "where am I?" For example, running pwd might show `/home/student/projects`.

**ls** lists files and folders in your current location. Use `ls -la` to see hidden files and detailed information like permissions and file sizes.

**cd** changes directories. Use `cd /var/log` to jump to the logs folder, or `cd ..` to go up one level.



# Working with Files & Directories

1

## Create Directories

`mkdir project_files` creates a new folder. Use `mkdir -p documents/work/reports` to create nested directories all at once.

2

## Create Files

`touch index.html` creates an empty file. It's perfect for quickly setting up file structures before adding content.

3

## Copy Items

`cp config.txt backup_config.txt` duplicates files. Use `cp -r folder1 folder2` to copy entire directories with all contents.

4

## Move & Rename

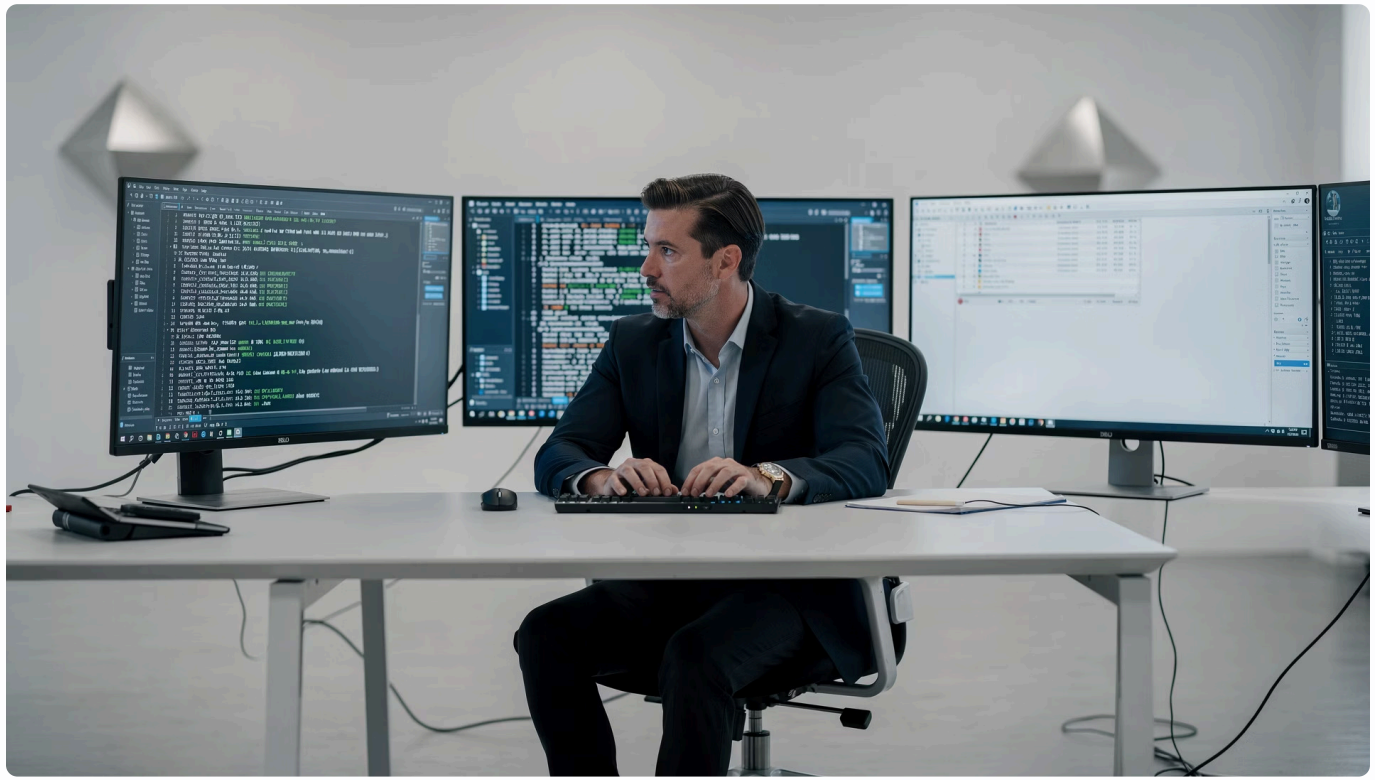
`mv oldname.txt newname.txt` renames files. Or `mv file.txt /home/user/documents/` moves it to a different location.

5

## Remove Items

`rm unwanted.txt` deletes files. Use `rm -rf old_project/` to remove directories, but be careful—there's no undo!

# Viewing & Searching File Content



## Read Files Effectively

**cat** displays entire file contents. Use `cat error.log` to quickly view small files.

**less** opens large files for scrolling. Navigate with arrow keys and press **q** to quit. Perfect for reviewing lengthy log files.

**head** and **tail** show the first or last 10 lines. Use `tail -f /var/log/syslog` to watch logs update in real-time—essential for troubleshooting.

**grep** searches for patterns. For example, `grep "error" app.log` finds all lines containing "error" in your application logs.

# Real-World Command Example

### Scenario: Finding Recent Errors

Your web application is acting strange. Users report intermittent failures. Here's how you'd investigate using Linux commands:

```
# Navigate to the log directory
cd /var/log/nginx

# List log files sorted by modification time
ls -lt

# Search for error messages in the access log
grep "500" access.log | tail -20

# Watch errors happen in real-time
tail -f error.log
```

This workflow helps you quickly identify issues. The `grep` command finds HTTP 500 errors, while `tail -f` shows new errors as they occur. These simple commands can save hours of troubleshooting time.

CHAPTER 3

# User & Group Management

## Understanding Users in Linux

Linux is a multi-user system. Each person (or application) needs their own account with specific permissions. This separation ensures security and prevents accidental damage to system files.

Think of users like employees in a company—each has different access levels. Some can only read documents, while administrators can modify anything. Groups work like departments, making it easy to assign permissions to multiple users at once.

Managing users properly is critical for security. Never run services as root, and always follow the principle of least privilege—give users only the access they actually need.

# User Management Commands

## useradd

Creates new user accounts

`sudo useradd -m -s /bin/bash john`

The `-m` flag creates a home directory, and `-s` sets their default shell.

## passwd

Sets or changes passwords

`sudo passwd john`

Always use strong passwords with mixed characters, numbers, and symbols.

## usermod

Modifies user properties

`sudo usermod -aG sudo john`

This adds John to the sudo group, granting administrative privileges.

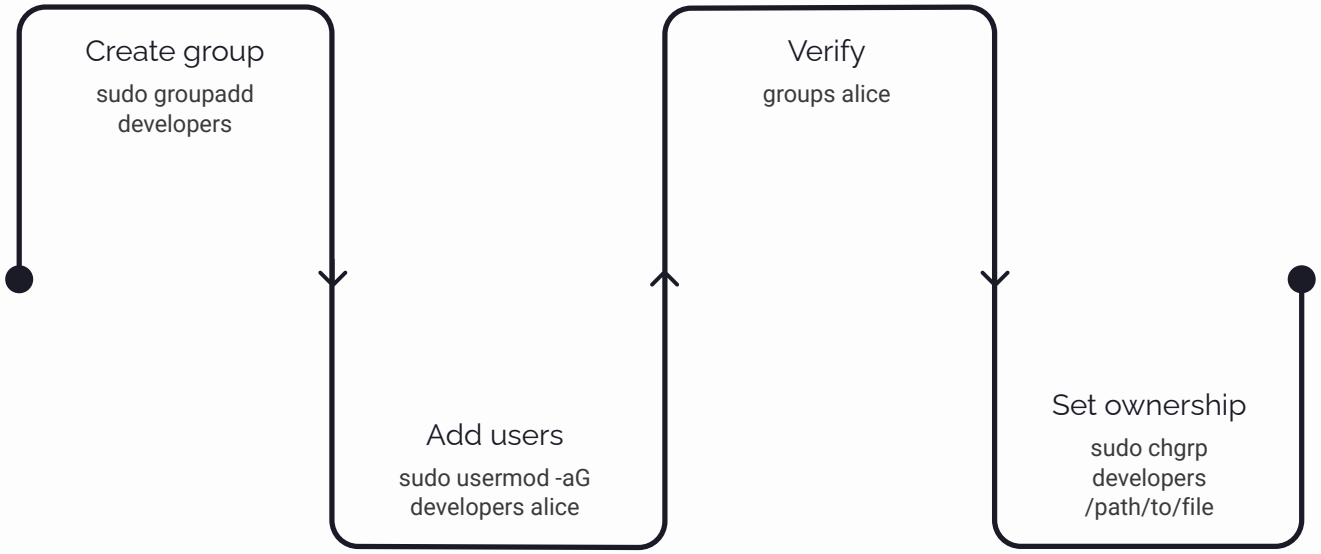
## userdel

Removes user accounts

`sudo userdel -r john`

The `-r` flag also deletes their home directory and files.

# Working with Groups




Groups simplify permission management. Instead of setting permissions for each user individually, assign them to a group and manage the group permissions.

## Essential Group Commands

**groupadd:** `sudo groupadd developers` creates a new group for your development team.

**usermod -aG:** `sudo usermod -aG developers sarah` adds Sarah to the developers group without removing her from other groups.

**groups:** `groups sarah` displays all groups Sarah belongs to.

**chgrp:** `chgrp developers /opt/project` changes the group ownership of a directory, allowing all developers to access it.

# Practical User Management Scenario

## Setting Up a Development Team

You're tasked with onboarding three new developers who need access to shared project files. Here's the complete workflow:

```
# Create the developers group
sudo groupadd developers

# Create user accounts with home directories
sudo useradd -m -s /bin/bash alice
sudo useradd -m -s /bin/bash bob
sudo useradd -m -s /bin/bash charlie

# Set passwords (you'll be prompted to enter them)
sudo passwd alice
sudo passwd bob
sudo passwd charlie

# Add users to the developers group
sudo usermod -aG developers alice
sudo usermod -aG developers bob
sudo usermod -aG developers charlie

# Create shared project directory
sudo mkdir /opt/projects
sudo chgrp developers /opt/projects
sudo chmod 770 /opt/projects
```

Now Alice, Bob, and Charlie can all access and modify files in /opt/projects, but other users cannot. This setup demonstrates proper security and collaboration practices.


CHAPTER 4

# File Access Permissions

Linux permissions control who can read, write, or execute files. This three-layer security system protects sensitive data and prevents unauthorized changes. Every file has permissions for the owner, the group, and everyone else. Understanding this system is fundamental to Linux security.

# Understanding Permission Notation

## Reading Permission Strings

When you run `ls -l`, you see permissions like `-rwxr-xr--`. Let's decode this:

The first character shows the type (- for file, d for directory). The next nine characters are grouped in threes:

- **Owner permissions (rwx):** The file owner can read, write, and execute
- **Group permissions (r-x):** Group members can read and execute, but not modify
- **Other permissions (r--):** Everyone else can only read the file

Think of **r** as viewing, **w** as editing, and **x** as running. A dash (-) means that permission is denied.

# Numeric Permission System

| 4 | 2 | 1 |
| --- | --- | --- |
| Read | Write | Execute |
| Permission to view file contents or list directory contents | Permission to modify files or create/delete files in directories | Permission to run files as programs or enter directories |

Linux uses octal (base-8) numbers for permissions. Add the values together to create permission combinations. For example, **7** (4+2+1) means read, write, and execute. **6** (4+2) means read and write only. **5** (4+1) means read and execute only.

The command `chmod 755 script.sh` sets owner to 7 (rwx), group to 5 (r-x), and others to 5 (r-x). This is perfect for scripts that everyone can run but only the owner can modify.

# Common Permission Patterns

### 644 - Standard File

`-rw-r--r--`

Owner can read and write, everyone else can only read. Perfect for documents and configuration files.

### 755 - Executable

`-rwxr-xr-x`

Owner has full control, others can read and execute. Use for scripts and programs.

### 700 - Private File

`-rwx------`

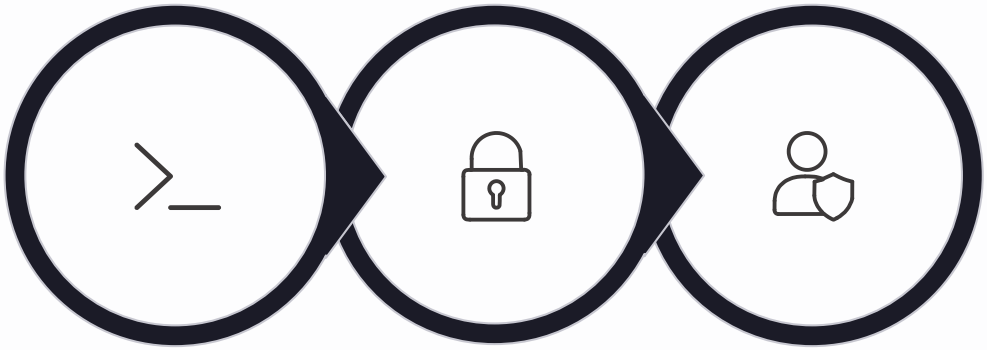
Only the owner has access. Ideal for sensitive data like SSH keys or password files.

### 777 - Fully Open

`-rwxrwxrwx`

Everyone can do anything. Avoid this—it's a major security risk except in rare debugging scenarios.

# Changing Permissions & Ownership




Properly managing permissions prevents security vulnerabilities and ensures smooth collaboration between users.

## Key Commands

**chmod:** Changes file permissions. Use `chmod 640 database.conf` to make a config file readable by owner and group, but hidden from others.

**chown:** Changes file ownership. `sudo chown www-data:www-data /var/www/html/index.html` gives your web server control of a web page.

**Symbolic mode:** Use letters instead of numbers. `chmod u+x script.sh` adds execute permission for the user. `chmod g-w file.txt` removes write permission for the group.


CHAPTER 5

# Special Permissions

Beyond standard read, write, and execute permissions, Linux offers three special permission types that provide advanced security controls. These permissions solve specific problems related to privilege escalation and shared directory management.

# SUID - Set User ID


## Temporary Privilege Elevation

SUID allows users to run a program with the permissions of the file owner. The most common example is the `passwd` command, which needs to modify system files that regular users can't access.

When you set SUID on a file, it appears as `-rwsr-xr-x`. Notice the **s** in the owner's execute position. Set it with `chmod 4755 filename` or `chmod u+s filename`.

**Security Warning:** SUID on scripts owned by root can be extremely dangerous. Attackers could exploit poorly written SUID programs to gain root access. Only use SUID when absolutely necessary.

# SGID - Set Group ID

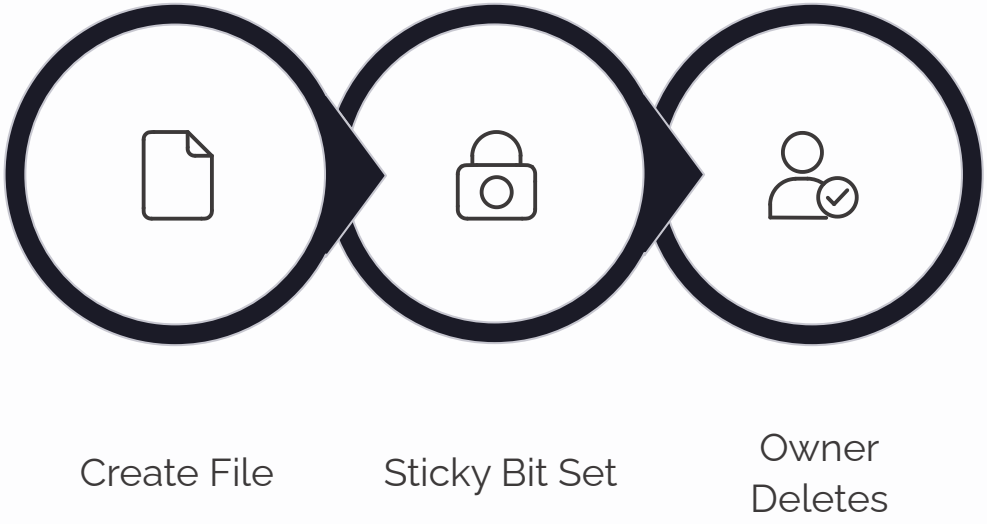
## Inherited Group Ownership

SGID has two important uses. On executable files, it works like SUID but with group permissions. More commonly, SGID on directories ensures all new files inherit the directory's group ownership.

For example, when developers create files in a shared project directory with SGID set, those files automatically belong to the "developers" group—even if the user's primary group is different.

Set SGID with `chmod 2755 directoryname` or `chmod g+s directoryname`. It appears as `drwxr-sr-x` with an **s** in the group's execute position.

# Sticky Bit




The sticky bit is essential for shared directories where multiple users store files.

## Protecting Shared Spaces

The sticky bit prevents users from deleting files they don't own, even if they have write permission on the directory. The `/tmp` directory uses this by default.

Without the sticky bit, any user with write access to a directory could delete anyone else's files. With the sticky bit set, you can only delete your own files—perfect for shared workspaces.

Set it with `chmod 1777 shared_folder` or `chmod +t shared_folder`. It appears as `drwxrwxrwt` with a **t** in the others' execute position.

# Default Permissions with umask

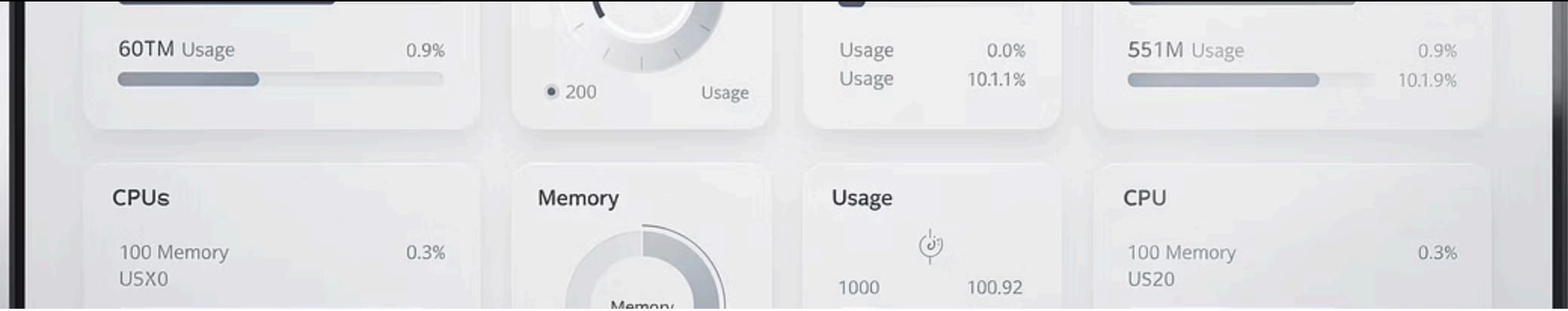
## Setting Creation Defaults

Every time you create a file or directory, Linux applies a default permission mask called **umask**. This mask determines what permissions are *removed* from the maximum possible permissions.

The default maximum permissions are 666 for files (rw-rw-rw-) and 777 for directories (rwxrwxrwx). The umask value is subtracted from these maximums.

A common umask of **0022** removes write permission for group and others, resulting in 644 for files (rw-r--r--) and 755 for directories (rwxr-xr-x). This provides a good security baseline.

Check your current umask with the `umask` command. Change it temporarily with `umask 0027`, or permanently by adding it to your bash profile.




CHAPTER 6

# Process & Service Management

Processes are running programs on your system. Managing them effectively means understanding how to start, stop, monitor, and control services. Whether troubleshooting a frozen application or managing server daemons, these skills are essential for every sysadmin.

# Viewing & Monitoring Processes

## ps Command

Shows currently running processes. Use `ps aux` to see all processes with detailed information including CPU and memory usage.

## top Command

Provides a live, updating view of system processes. Press **q** to quit, **k** to kill a process, or **M** to sort by memory usage.

## htop Command

An improved version of top with a colorful, interactive interface. Navigate with arrow keys, **F9** to kill processes, and **F10** to quit.

## pgrep Command

Searches for processes by name. `pgrep nginx` returns process IDs for all nginx processes, useful for scripts and automation.

# Controlling Processes

## Starting & Stopping

**kill:** Sends signals to processes. `kill -15 1234` gracefully stops process 1234, giving it time to clean up. `kill -9 1234` immediately terminates it—use this as a last resort.

**killall:** Terminates all processes by name. `killall firefox` closes all Firefox instances at once.

**pkill:** Similar to killall but with pattern matching. `pkill -u john` terminates all processes owned by user John.

**Background jobs:** Add `&` to run commands in the background. `long_script.sh &` lets you continue using the terminal while it runs.

# Managing Services with systemd

01

## Check Status

`systemctl status nginx` shows whether the service is running, recent log entries, and when it started.

02

## Start Service

`sudo systemctl start nginx` launches the service immediately but doesn't enable it at boot.

03

## Enable at Boot

`sudo systemctl enable nginx` configures the service to start automatically when the system boots.

04

## Restart Service

`sudo systemctl restart nginx` stops and starts the service—useful after configuration changes.

05

## Stop Service

`sudo systemctl stop nginx` shuts down the service gracefully until manually started again.

Modern Linux distributions use systemd for service management. These commands replace older init.d scripts and provide better logging, dependency management, and control over system services.

CHAPTER 7

# Introduction to Bash

## Your Command Line Environment

Bash (Bourne Again Shell) is the default command interpreter on most Linux systems. It's both an interactive command-line interface and a powerful scripting language.

When you log in, Bash reads configuration files that customize your environment—setting your prompt, defining shortcuts (aliases), and configuring your PATH variable. Understanding these files lets you personalize your workspace and boost productivity.

Think of Bash as your control center. Every command you type, every script you run, passes through Bash. Mastering it transforms you from a casual user into a power user.

# Key Bash Configuration Files

1

## .bash_profile

Runs once when you log in. Set environment variables like PATH, and configure system-wide settings here. Often sources .bashrc automatically.

2

## .bashrc

Executes every time you open a new terminal window. Define aliases, functions, and prompt customizations here. This is your primary customization file.

3

## .bash_history

Stores your command history—usually the last 500-1000 commands. Press the up arrow to recall previous commands. Use `history` to view the full list.

4

## .bash_logout

Runs when you log out of a login shell. Use it to clean up temporary files, archive logs, or display goodbye messages.

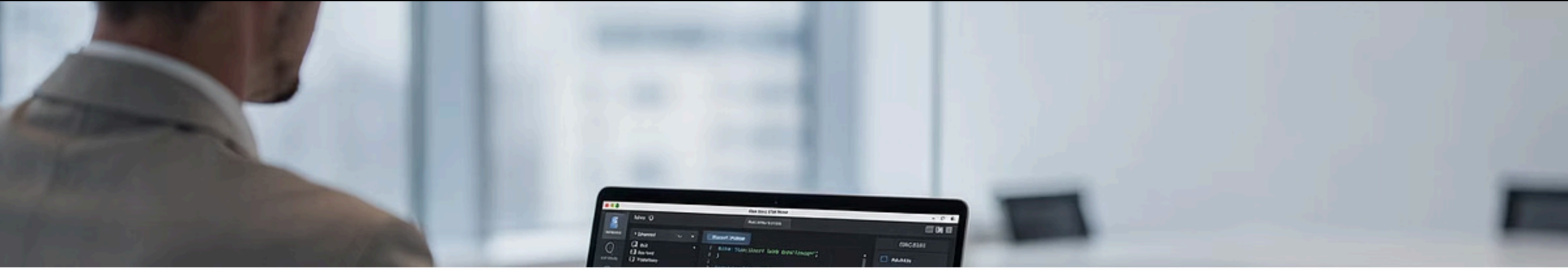
# Customizing Your Bash Environment

### Sample .bashrc Customizations

Here are practical additions to make your command line more efficient and user-friendly:

```
# Colorful prompt showing username, host, and current directory
export PS1="\[\e[32m\]\u@\h\[\e[m\]:\[\e[34m\]\w\[\e[m\]\$ "

# Useful aliases for common commands
alias ll='ls -alFh'
alias update='sudo apt update && sudo apt upgrade'
alias ports='netstat -tulanp'
alias grep='grep --color=auto'

# Custom function to quickly create and enter directories
mkcd() {
    mkdir -p "$1" && cd "$1"
}

# Enable command history search with arrow keys
bind '"\e[A": history-search-backward'
bind '"\e[B": history-search-forward'

# Add custom scripts directory to PATH
export PATH="$HOME/scripts:$PATH"
```

After editing .bashrc, run `source ~/.bashrc` to apply changes immediately without logging out.



# Your Linux Journey Continues

### Practice Daily

The only way to master Linux is through hands-on experience. Create your own test environment and experiment without fear.

### Build Real Projects

Set up a web server, automate backups with scripts, or configure a development environment. Real challenges teach real skills.

### Join the Community

Linux forums, Reddit communities, and local user groups offer support and knowledge. Never hesitate to ask questions—everyone started where you are.

You now have the foundational knowledge to manage users, control permissions, monitor processes, and customize your environment. These skills form the bedrock of Linux system administration. Keep exploring, keep learning, and remember—every expert was once a beginner who refused to give up.