Mastering Linux & Cloud: A Hands-On Journey

Welcome to your comprehensive guide to Linux system administration and cloud computing fundamentals. This presentation will take you from basic commands to advanced cloud concepts, equipping you with practical skills that modern IT environments demand.

CHAPTER 1

Introduction to Linux and Distributions

Linux is an open-source operating system kernel that powers everything from smartphones to supercomputers. Unlike Windows or macOS, Linux comes in many "flavors" called distributions (distros), each tailored for different use cases.

Popular distributions include Ubuntu (beginner-friendly), CentOS/RHEL (enterprise), Debian (stable), and Arch (customizable). Each distro uses the same Linux kernel but packages different software and tools.

Think of it like cars: all have engines, but a Toyota Camry serves different needs than a Ford F-150 truck.

Why Learn Linux?

- Powers 90% of cloud infrastructure
- Free and open-source
- Essential for DevOps careers
- Highly customizable
- Strong security model

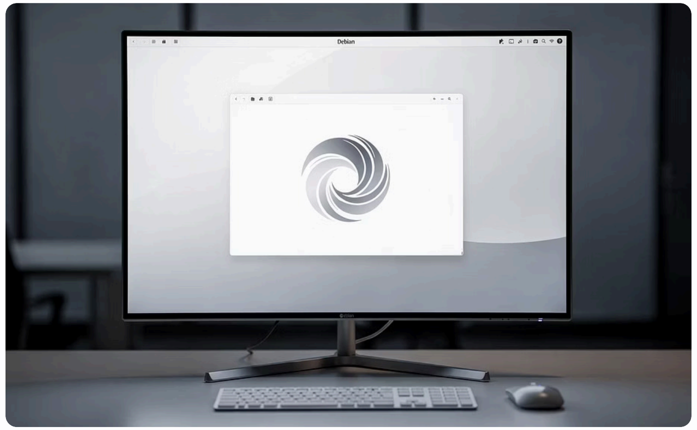
Popular Linux Distributions Compared

Ubuntu

Perfect for beginners. User-friendly interface, massive community support, and extensive documentation. Great for learning and desktop use.

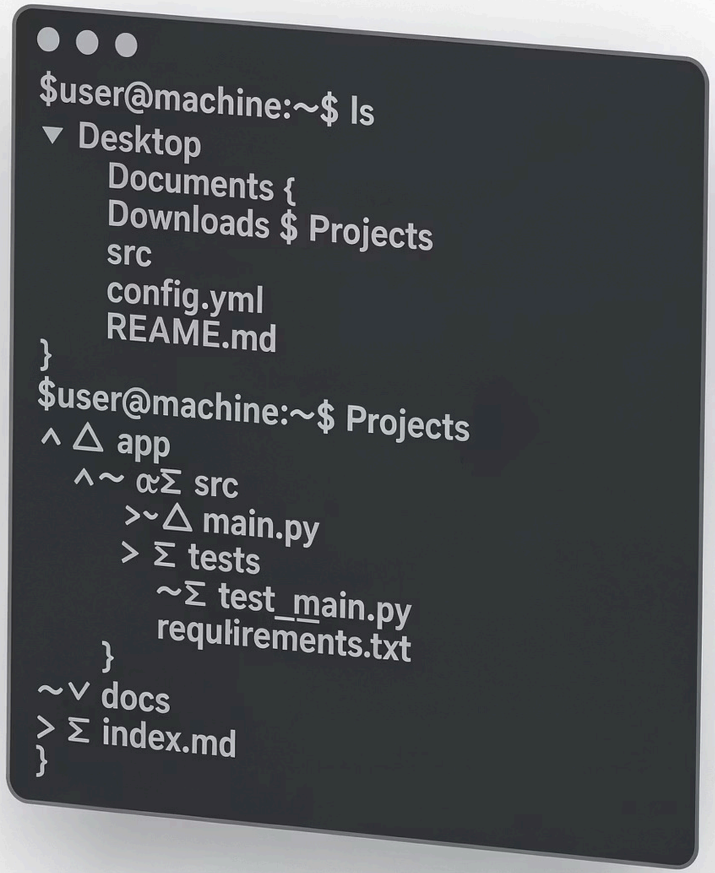
RHEL/CentOS

Enterprise standard. Rock-solid stability, long-term support, and preferred by corporations. CentOS is the free version of RHEL.



Debian

The foundation of Ubuntu. Ultra-stable, security-focused, and respected in server environments. Favored by system administrators.



```
$user@machine:~$ ls
▼ Desktop
    Documents {
    Downloads $ Projects
    src
    config.yml
    REAME.md
}
$user@machine:~$ Projects
^ △ app
  ^~ αΣ src
     >~△ main.py
     > Σ tests
       ~Σ test_main.py
       requlirements.txt
   }
~∨ docs
> Σ index.md
}
```

CHAPTER 2

Essential Linux Commands & File System Navigation

The Linux command line is your primary interface for system administration. Unlike graphical interfaces, the terminal gives you direct, powerful control over the system. Mastering basic navigation commands is the foundation of Linux proficiency.

The Linux file system is organized as a tree structure, starting from the root directory (/) and branching into subdirectories. Understanding this hierarchy is crucial for effective system navigation.

Core Navigation Commands

pwd

Print Working Directory - shows your current location in the file system

```
$ pwd
/home/student/projects
```

ls

List directory contents. Use ls -la for detailed view including hidden files

```
$ ls -la
drwxr-xr-x 5 user
```

cd

Change Directory - navigate between folders

```
$ cd /var/log
$ cd .. (go up one level)
```

File Manipulation Commands

Creating & Viewing Files

```
$ touch newfile.txt
$ cat file.txt
$ less largefile.log
$ head -n 10 file.txt
$ tail -f /var/log/syslog
```

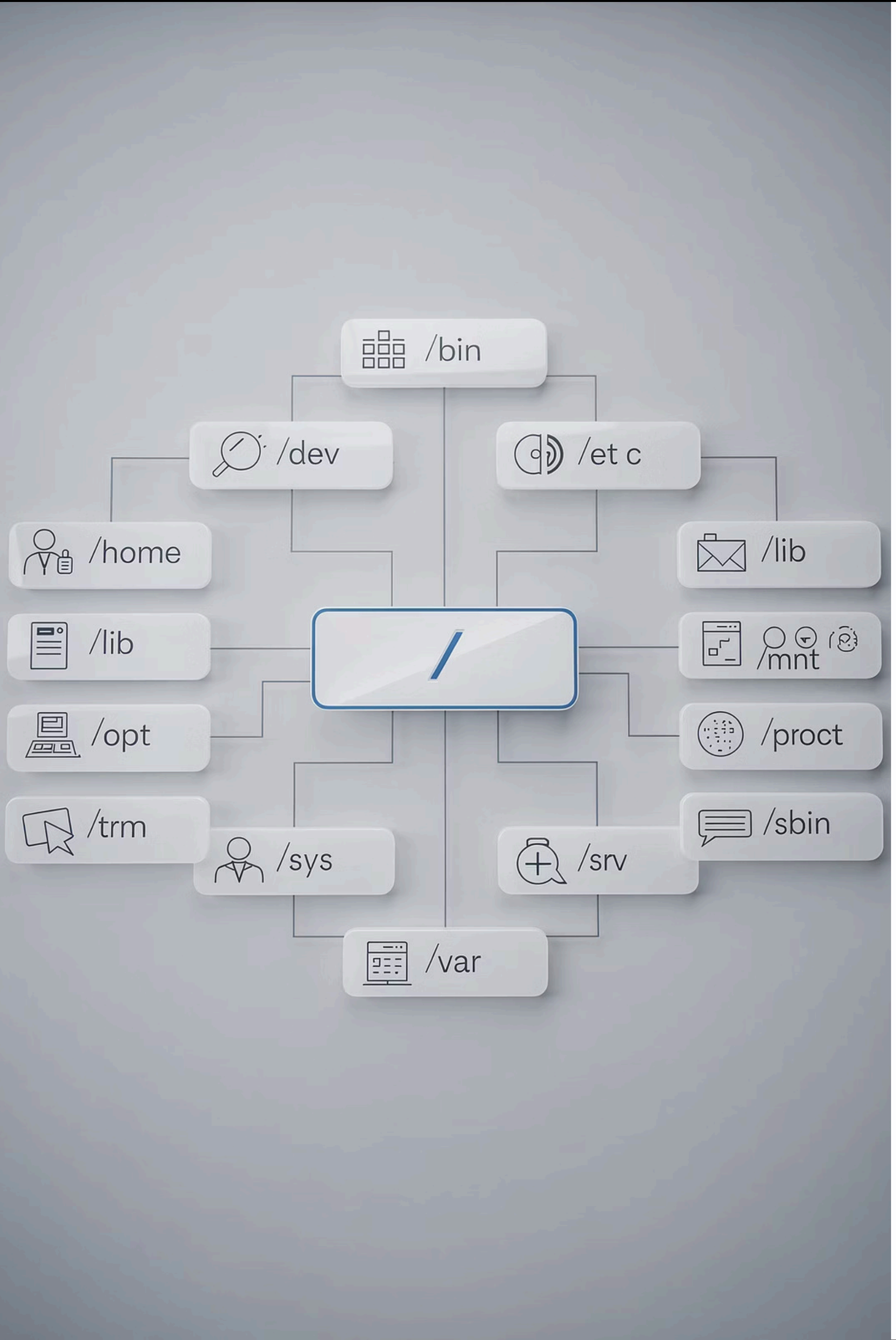
The **cat** command displays entire files, while **less** allows scrolling through large files. The **tail -f** command is invaluable for monitoring log files in real-time.

Copying & Moving

```
$ cp source.txt destination.txt
$ mv oldname.txt newname.txt
$ rm unwanted.txt
$ mkdir new_directory
$ rmdir empty_directory
```

Always use **rm -i** for interactive deletion to prevent accidents. The **mv** command serves double duty: moving files and renaming them.

Understanding the Linux File System Hierarchy



/home

User home directories. Your personal files live here (e.g., /home/john).

/etc

System configuration files. Critical settings for services and applications.

/var

Variable data like logs, databases, and web content that changes frequently.

/usr

User programs and applications. Most software installs here.

CHAPTER 3

User and Group Management

Linux is a multi-user operating system, designed from the ground up to allow multiple people to use the same computer simultaneously and securely. Understanding user and group management is essential for system security and access control.

Every user has a unique User ID (UID), and every group has a Group ID (GID). These numerical identifiers are what Linux actually uses internally, while usernames are just human-friendly labels.

The /etc/passwd File

This critical file stores user account information. Each line represents one user, with seven colon-separated fields. Let's decode a real example:

```
john:x:1001:1001:John Doe:/home/john:/bin/bash
```

- **john** - Username
- **x** - Password placeholder (actual password in /etc/shadow)
- **1001** - User ID (UID)
- **1001** - Primary Group ID (GID)
- **John Doe** - Full name (GECOS field)
- **/home/john** - Home directory
- **/bin/bash** - Default shell

Key Commands

```
$ cat /etc/passwd
$ useradd newuser
$ usermod -aG groupname user
$ userdel username
$ passwd username
```

Regular users have UIDs starting from 1000, while system accounts use lower numbers (0-999). Root always has UID 0.

The /etc/group File

Groups provide a way to organize users and control access to resources collectively. Instead of granting permissions to individual users, you assign them to a group. This file defines all groups on the system.

```
developers:x:1005:john,sarah,mike
webadmin:x:1006:john,lisa
```

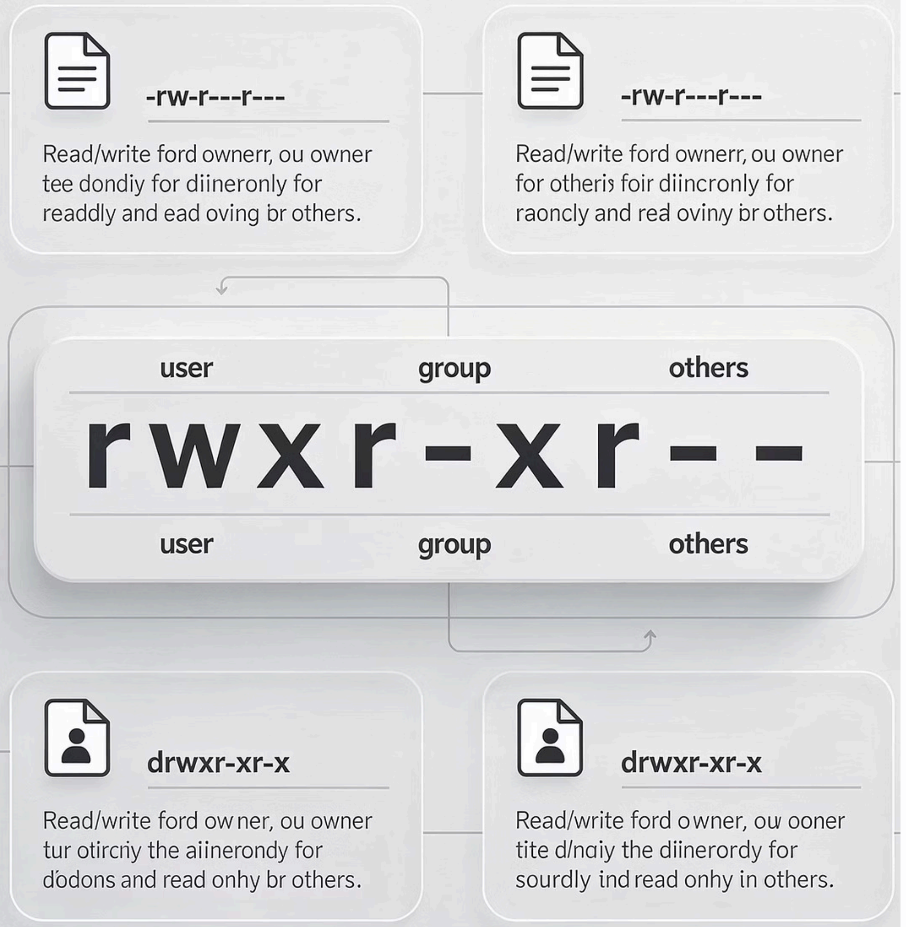
Format: **groupname:password:GID:members**

In this example, john, sarah, and mike are all in the developers group (GID 1005). John is also in the webadmin group, giving him access to resources both groups can access.

Group Commands

```
$ groupadd newgroup
$ groupdel oldgroup
$ groups username
$ id username
```

The **id** command shows all groups a user belongs to, which is helpful for troubleshooting permission issues.


-rw-r---r---
Read/write ford ownerr, ou owner tee dondiy for diineronly for readdly and ead oving br others.
-rw-r---r---
Read/write ford ownerr, ou owner for otheris foir diincronly for raoncly and real oviny br others.
user
group
others
rwxr-xr--
user
group
others
drwxr-xr-x
Read/write ford ow ner, ou owner tur otircriy the aiinerondy for dodons and read onhy br others.
drwxr-xr-x
Read/write ford owner, ou ooner tite d/naiy the diinerordy for sourdly ind read onhy in others.

CHAPTER 4

File and Directory Permissions

Linux permissions control who can read, write, or execute files and directories. This three-tier system (owner, group, others) is the foundation of Linux security. Understanding permissions prevents unauthorized access and accidental file modifications.

Every file and directory has three permission sets: one for the owner, one for the group, and one for everyone else. Each set contains three permission types: read (r), write (w), and execute (x).

Reading Permission Notation

```
-rwxr-xr-- 1 john developers 4096
```

Let's break this down character by character:

- **-** File type (- = file, d = directory, l = link)
- **rwx** Owner permissions (read, write, execute)
- **r-x** Group permissions (read, execute only)
- **r--** Others permissions (read only)

This file is fully accessible to john, executable by the developers group, and readable by everyone else.

Numeric Notation

Permissions can also be expressed as three-digit numbers:

- **r** = 4 (read)
- **w** = 2 (write)
- **x** = 1 (execute)

Add them up for each set:

```
rwx = 7 (4+2+1)
r-x = 5 (4+0+1)
r-- = 4 (4+0+0)
```

So rwxr-xr-- = **754**

Changing Permissions and Ownership

chmod

Change file permissions

```
$ chmod 755 script.sh
$ chmod u+x file.txt
$ chmod g-w document.txt
```

chown

Change file owner

```
$ chown john file.txt
$ chown john:developers app/
$ chown -R user:group dir/
```

chgrp

Change group ownership

```
$ chgrp developers project/
$ chgrp -R webteam site/
```

The **-R** flag applies changes recursively to all files and subdirectories. Always be cautious when using recursive operations, especially with system directories.

Real-World Permission Examples

Web Server Files

```
chmod 644 index.html
```

Owner can edit, everyone can read. Perfect for public web content that only the owner should modify.

Executable Scripts

```
chmod 755 backup.sh
```

Owner has full control, others can run but not modify. Standard for system scripts and executables.

Private Files

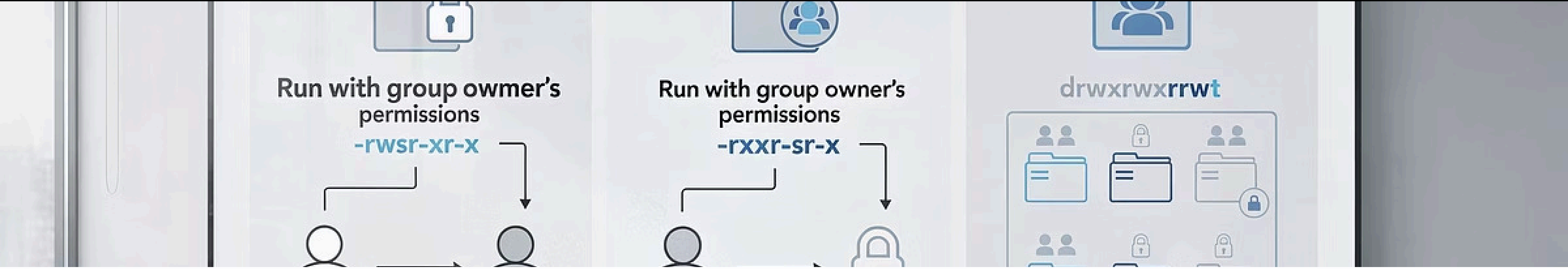
```
chmod 600 credentials.txt
```

Only owner can read/write. Critical for sensitive data like passwords and API keys.

Shared Project Directory

```
chmod 770 /project
```

Owner and group have full access, others locked out. Ideal for team collaboration directories.


Run with group owmer's permissions
-rwsr-xr-x
Run with group owner's permissions
-rxxr-sr-x
drwxrwxrrwt

CHAPTER 5

SUID, SGID, Sticky Bit, and umask

Beyond standard permissions, Linux offers special permission bits that provide fine-grained control over file execution and directory behavior. These advanced features are crucial for security-sensitive environments and multi-user systems.

SUID (Set User ID)

When SUID is set on an executable file, it runs with the permissions of the file owner, not the user executing it. This is essential for programs that need elevated privileges.

The classic example is the **passwd** command. Regular users need to change their passwords, but password data is stored in protected files. SUID allows passwd to run as root temporarily.

```
-rwsr-xr-x 1 root root /usr/bin/passwd
```

Notice the **s** in place of the owner's execute bit.

Setting SUID

```
$ chmod u+s program
$ chmod 4755 program
```

The **4** in numeric notation represents SUID.

Security Warning: SUID files are powerful but dangerous. A vulnerable SUID root program can grant attackers full system access. Only apply SUID when absolutely necessary.

SGID (Set Group ID) and Sticky Bit

SGID on Files

Similar to SUID but inherits group permissions instead. The file runs with the group's privileges.

```
$ chmod g+s file
$ chmod 2755 file
```

Indicated by **s** in group execute position: **-rwxr-sr-x**

SGID on Directories

When set on a directory, new files inherit the directory's group, not the creator's primary group. Perfect for shared project folders.

```
$ chmod g+s /shared_project
drwxrwsr-x 2 john developers
```

All files created inside get **developers** as their group.

Sticky Bit

On directories, only the file owner (or root) can delete files, even if others have write permission. Essential for /tmp directory.

```
$ chmod +t directory
$ chmod 1777 /tmp
drwxrwxrwt 15 root root
```

Notice the **t** at the end. Prevents users from deleting each other's temporary files.

umask: Default Permission Control

The **umask** command sets default permissions for newly created files and directories. It works by subtracting permissions from the maximum allowable: 666 for files, 777 for directories.

A umask of 022 means:

- Files: 666 - 022 = **644** (rw-r--r--)
- Directories: 777 - 022 = **755** (rwxr-xr-x)

Common umask values:

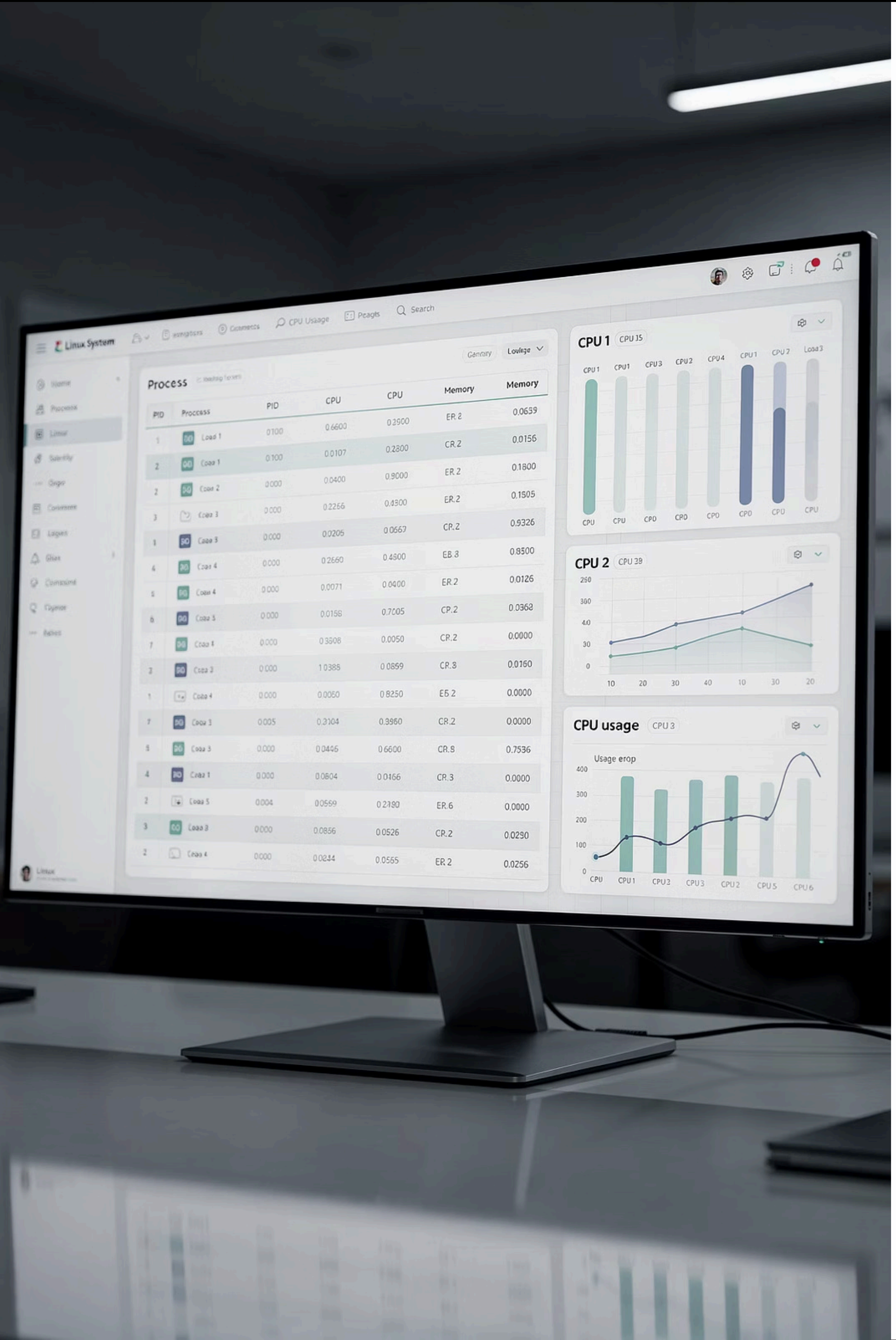
- **022** - Standard (owner full control, others read)
- **002** - Group-friendly (group can write)
- **077** - Private (only owner has any access)

Using umask

```
$ umask
0022
$ umask 077
$ touch newfile.txt
$ ls -l newfile.txt
-rw------- 1 user user
```

Set permanently in **~/.bashrc**:

```
umask 022
```



CHAPTER 6

Process and Service Management

Processes are running instances of programs. Managing processes effectively is crucial for system performance, troubleshooting, and maintaining service availability. Linux provides powerful tools to monitor, control, and automate process execution.

Monitoring Processes

ps Command

Static snapshot of current processes. Essential for quick process inspection.

```
$ ps aux
$ ps -ef | grep nginx
$ ps -u username
```

Shows PID (Process ID), CPU/memory usage, and command details. The **aux** flags show all processes with detailed info.

top Command

Real-time, dynamic view of system processes, refreshing every few seconds.

```
$ top
$ top -u john
```

Interactive interface shows CPU, memory, and swap usage. Press **q** to quit, **k** to kill a process, **M** to sort by memory.

htop Command

Enhanced version of top with colorful, user-friendly interface and better navigation.

```
$ htop
```

Easier to read, allows mouse interaction, displays process trees visually. Must be installed separately on most systems. Press **F9** to kill processes.

Killing Processes

Sometimes processes freeze, consume excessive resources, or need to be stopped. The **kill** command sends signals to processes, instructing them to terminate or modify behavior.

Common Signals

- **SIGTERM (15)** - Polite request to stop (default)
- **SIGKILL (9)** - Force immediate termination
- **SIGHUP (1)** - Reload configuration
- **SIGSTOP (19)** - Pause process

Always try SIGTERM first. SIGKILL should be a last resort as it doesn't allow cleanup.

Examples

```
$ kill 1234
$ kill -9 1234
$ kill -SIGTERM 1234
$ killall firefox
$ pkill -u username
```

killall terminates by process name, while **pkill** offers more flexible pattern matching. Use carefully in production environments.

Service Management with systemctl

Modern Linux systems use **systemd** to manage services (daemons). The **systemctl** command provides complete control over starting, stopping, and configuring services to run at boot.

01

Check Service Status

```
$ systemctl status nginx
```

Shows whether service is running, recent log entries, and process details.

02

Start/Stop Services

```
$ sudo systemctl start nginx
$ sudo systemctl stop nginx
$ sudo systemctl restart nginx
```

Immediate effect but not persistent across reboots.

03

Enable at Boot

```
$ sudo systemctl enable nginx
$ sudo systemctl disable nginx
```

Ensures service starts automatically when system boots.

04

View All Services

```
$ systemctl list-units --type=service
$ systemctl list-unit-files
```

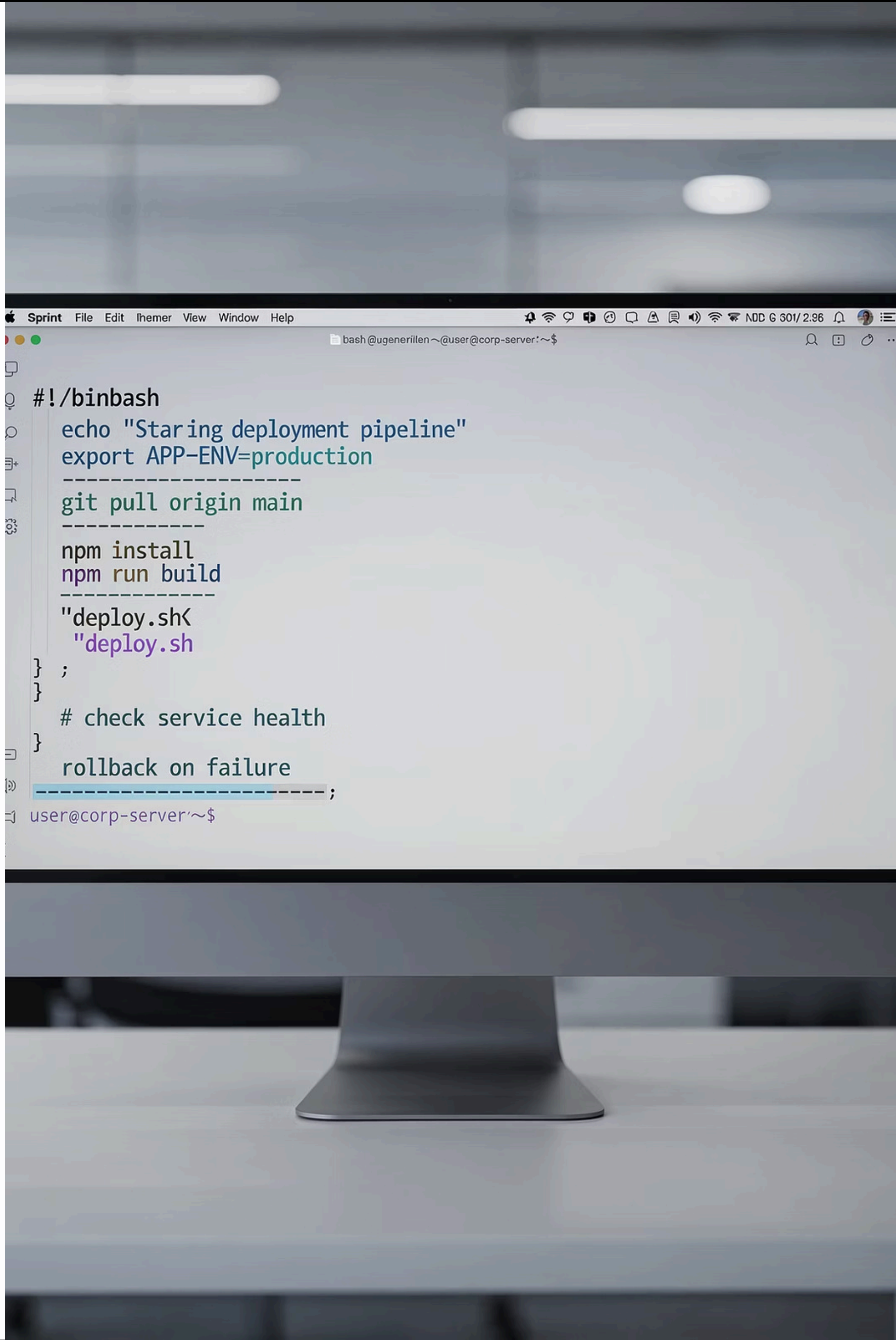
Comprehensive overview of all available services and their states.

CHAPTER 7

Bash Basics

Bash (Bourne Again Shell) is the default command-line interpreter on most Linux systems. It's not just for typing commands—it's a full-fledged programming language for automating tasks, managing systems, and building powerful workflows.

Learning Bash scripting transforms you from a command user to a system administrator who can automate repetitive tasks and build sophisticated automation solutions.


Sprint File Edit Ihemer View Window Help
NDD G 301/ 2:96
bash @ugenerillen~@user@corp-server:~$

```
#!/binbash
  echo "Staring deployment pipeline"
  export APP-ENV=production
  --------------------
  git pull origin main
  ------------
  npm install
  npm run build
  -------------
  "deploy.sh<
   "deploy.sh
} ;
}
  # check service health
}
  rollback on failure
-------------------------;
user@corp-server~$
```

Important Bash Configuration Files

1 .bashrc

Executes for every new interactive shell. Place aliases, functions, and custom prompts here. Changes take effect immediately for new terminals.

```
alias ll='ls -la'
export PATH=$PATH:/custom/path
```

2 .bash_profile

Runs only once during login. Used for environment variables and startup programs. On many systems, sources .bashrc automatically.

```
export JAVA_HOME=/usr/lib/jvm/java-11
source ~/.bashrc
```

3 .bash_history

Stores command history. Access previous commands with up arrow or **history** command. Useful for reviewing past actions and debugging.

```
$ history | grep ssh
$ !456 (re-run command 456)
```

4 .bash_logout

Executes when you log out. Perfect for cleanup tasks, clearing sensitive data, or logging logout times.

```
clear
echo "Logged out at $(date)"
```

Conditionals, Loops, and Functions

If-Then-Else

```
if [ $age -ge 18 ]; then
  echo "Adult"
elif [ $age -ge 13 ]; then
  echo "Teen"
else
  echo "Child"
fi
```

Test conditions: **-eq** (equal), **-ne** (not equal), **-lt** (less than), **-gt** (greater than), **-f** (file exists).

For Loops

```
for i in {1..5}; do
  echo "Count: $i"
done

for file in *.txt; do
  mv "$file" "${file%.txt}.bak"
done
```

Iterate over sequences, arrays, or file lists. Powerful for batch operations.

Functions

```
backup() {
  tar -czf "backup-$(date
+%Y%m%d).tar.gz" "$1"
  echo "Backed up $1"
}

backup /home/user/docs
```

Reusable code blocks with parameters. Makes scripts modular and maintainable.

Debugging Bash Scripts

Debugging is essential when scripts don't behave as expected. Bash provides several mechanisms to trace execution and identify problems.

1

Echo Statements

Simple but effective. Print variable values at key points.

```
echo "DEBUG: username=$username"
echo "Processing file: $file"
```

2

Set -x Flag

Enables trace mode. Shows each command before execution.

```
#!/bin/bash
set -x
# Your script here
set +x  # disable tracing
```

3

Bash -x

Run entire script in debug mode from command line.

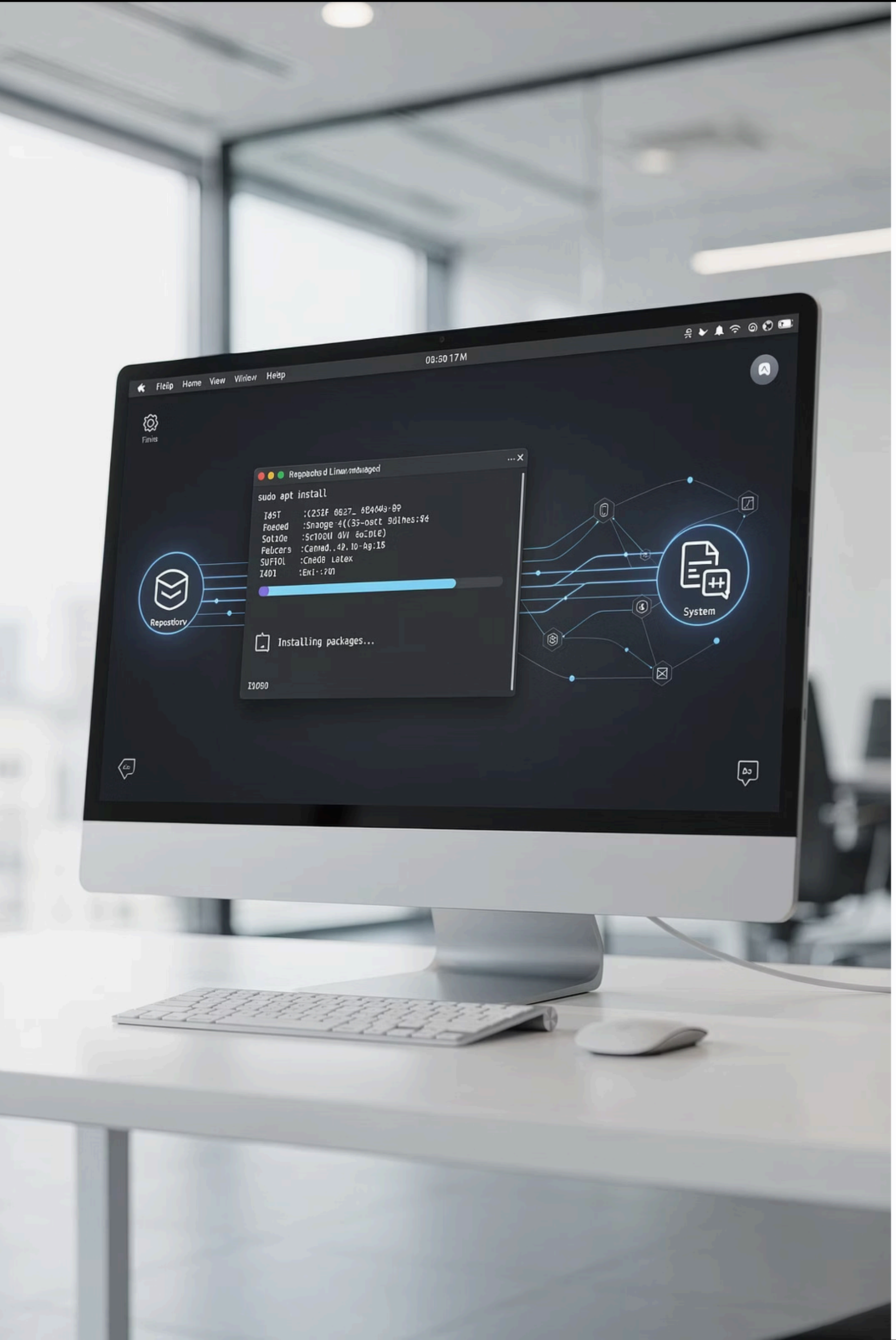
```
$ bash -x script.sh
```

4

Set -e Flag

Exit immediately if any command fails. Prevents cascading errors.

```
#!/bin/bash
set -e
```



CHAPTER 8

Package Management

Package managers automate software installation, updates, and removal. They handle dependencies, verify software integrity, and maintain system consistency. Different Linux distributions use different package managers, but the concepts remain similar.

APT, YUM, and DNF

APT (Debian/Ubuntu)

Advanced Package Tool for Debian-based systems. User-friendly and powerful.

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install nginx
$ sudo apt remove nginx
$ sudo apt search keyword
$ sudo apt autoremove
```

apt update refreshes package lists. **apt upgrade** installs updates. Always update before installing new packages.

YUM (CentOS/RHEL 7)

Yellowdog Updater Modified for Red Hat-based systems. Older but still widely used.

```
$ sudo yum update
$ sudo yum install httpd
$ sudo yum remove httpd
$ sudo yum search httpd
$ sudo yum list installed
```

Handles RPM packages. Being replaced by DNF in newer versions but still common in enterprise environments.

DNF (Fedora/RHEL 8+)

Dandified YUM. Modern replacement with better performance and cleaner dependency resolution.

```
$ sudo dnf update
$ sudo dnf install nginx
$ sudo dnf remove nginx
$ sudo dnf search nginx
$ sudo dnf history
```

Faster than YUM, better error messages, and improved memory usage. Commands nearly identical to YUM.

Snap and Flatpak: Universal Package Formats

Snap (by Canonical)

Self-contained packages that work across Linux distributions. Include all dependencies, ensuring consistency.

```
$ sudo snap install vlc
$ snap list
$ sudo snap remove vlc
$ sudo snap refresh
```

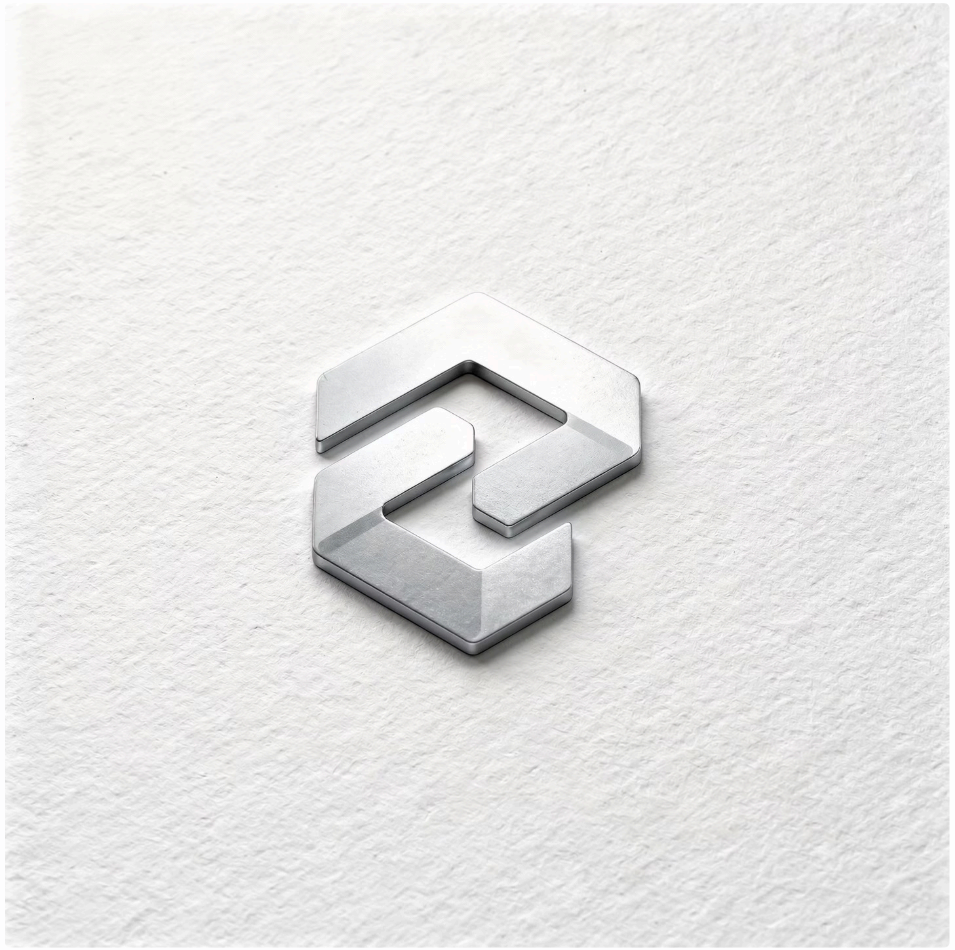
Snaps auto-update and run in isolated sandboxes for security. Popular for desktop applications. Slightly larger file sizes due to bundled dependencies.

Flatpak

Similar to Snap but with different architecture. More popular in GNOME-based distros.

```
$ flatpak install flathub org.gimp.GIMP
$ flatpak list
$ flatpak uninstall org.gimp.GIMP
$ flatpak update
```

Uses Flathub as main repository. Also sandboxed and distribution-agnostic. Preferred by some for better desktop integration.



Both Snap and Flatpak solve the "dependency hell" problem by packaging everything together. Traditional package managers (APT/YUM/DNF) remain faster and more efficient for system-level packages.

CHAPTER 9

Cloud Fundamentals from a DevOps Perspective

Cloud computing has revolutionized how we build and deploy applications. Understanding cloud fundamentals is essential for modern IT professionals, especially from a DevOps perspective where infrastructure is code and automation is key.

Cloud providers like AWS, Azure, and Google Cloud offer on-demand computing resources, eliminating the need for physical hardware management and enabling rapid scaling.