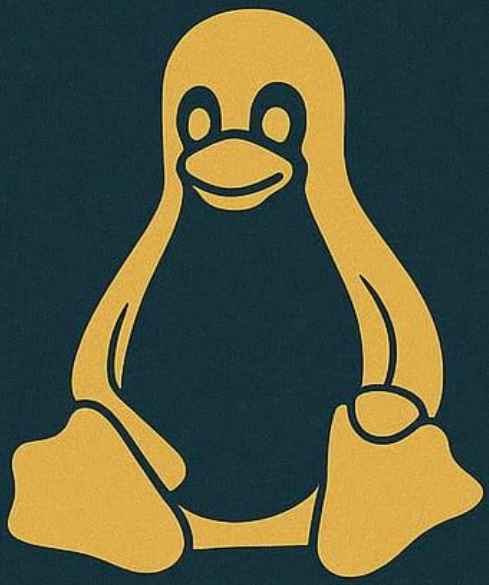
LINUX ESSENTIALS

A BEGINNER'S GUIDE TO THE COMMAND LINE AND BEYOND



FIRST EDITION

MD ROBAIDUL ISLAM ROBAEID

Table of Contents

🕮 Chapter 1: What is Linux?

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand what Linux is and where it came from.
- Learn why Linux matters in the modern computing world.
- Know the basic structure of a Linux system.
- Get introduced to distributions (distros) and how they differ.

1.1 What is Linux?

Linux is a free and open-source operating system that powers everything from web servers and smartphones to TVs and supercomputers. Unlike Windows or macOS, Linux gives users control over nearly every part of the system — and it's built by a global community of developers.

At its core, Linux is just the **kernel** — the central program that communicates between hardware and software. What most people call “Linux” (Ubuntu, Fedora, Arch, etc.) is a combination of the Linux kernel and a set of tools and applications, often referred to as a **Linux distribution**.

1.2 A Brief History

- **1969** – Unix is developed at Bell Labs. It's a powerful, multitasking OS, and the spiritual ancestor of Linux.
- **1983** – Richard Stallman launches the **GNU Project** to build a free Unix-like OS. GNU provides tools like compilers, shells, and editors — but not a kernel.
- **1991** – **Linus Torvalds**, a Finnish student, releases the first version of the **Linux kernel**. He posts it online, and the open-source community quickly begins contributing.

“Just a hobby, won’t be big and professional like GNU.”
— *Linus Torvalds*, 1991 Usenet post

The GNU tools and the Linux kernel combined to form what we now call a **Linux operating system**.

Note: GNU stands for **"GNU's Not Unix!"** — it’s a recursive acronym used for a free software project started by Richard Stallman. It means GNU is like Unix but completely free and open-source!

1.3 Why Linux?

- 🆓 **Free and Open Source**
  You can use, modify, and distribute it without cost or restrictions.
- 🔒 **Secure and Stable**
  Used in servers, Linux rarely crashes and is highly secure.
- ⚙️ **Customizable**
  You can change everything — from the desktop environment to how the system boots.
- 🌐 **Widely Used**
    - 90%+ of cloud servers run Linux
    - Android (a Linux-based OS) dominates the smartphone market
    - Supercomputers, routers, IoT devices — all run Linux

1.4 Components of a Linux System

Here's a quick breakdown of what makes up a typical Linux system:

Component	Description
Kernel	The core that interacts with hardware
Shell	Command-line interface (like `bash`)
User space	Programs and utilities run by users
File system	Organizes data on storage devices (`/`, `/home`, etc.)
Init system	Manages boot process and services (e.g., `systemd`)

1.5 What Is a Linux Distribution (Distro)?

A **distribution** is a complete Linux operating system package. It combines the kernel with utilities, package managers, desktop environments, and more.

Popular distros include:

Distro	Best For	Package Manager
Ubuntu	Beginners, general use	`apt`
Fedora	Developers, cutting-edge	`dnf`
Arch Linux	Advanced users, DIY	`pacman`
Debian	Stability	`apt`
Linux Mint	Easy desktop experience	`apt`
CentOS/RHEL	Servers, enterprise use	`yum`/`dnf`

1.6 Misconceptions About Linux

- **“Linux is only for hackers.”**
  Not true — it powers many user-friendly systems.
- **“You can’t run regular software.”**
  Many apps are cross-platform, and alternatives exist for most tasks.
- **“You need to use the terminal all the time.”**
  While the terminal is powerful, many distros offer full graphical interfaces.

Summary

- Linux is a free, open-source OS built around the Linux kernel.
- It’s used in everything from phones to supercomputers.
- The GNU project and Linux kernel together form the basis of modern Linux systems.
- Distros package the kernel and tools into user-friendly systems.
- Linux is powerful, flexible, and accessible — not just for geeks.

□ Exercises

1. In your own words, what is Linux?
2. Name two differences between Windows and Linux.
3. Pick a Linux distribution you're curious about. Look it up and write a few sentences about who it's for and what makes it unique.

🕮 Chapter 2: Installing Linux

🎯 Chapter Goals:

By the end of this chapter, you will:

- Know the safe ways to try Linux without wiping your current system.
- Understand how to install Linux on real hardware, virtual machines, or Windows.
- Learn the basics of the Linux filesystem structure.
- Get ready to start using Linux daily.

2.1 Try Before You Install

You don't need to wipe your existing system to try Linux! There are several ways to **test-drive Linux safely**:

🔹 Option 1: Live USB

A **Live USB** lets you boot into a full Linux system directly from a flash drive.

Steps:

1. **Download a Linux ISO** – e.g., Ubuntu from ubuntu.com
2. **Create a bootable USB** using:
    - **Rufus** (Windows)
    - **balenaEtcher** (macOS/Linux)
3. **Restart your computer**, boot from the USB drive, and choose "Try Ubuntu".

No changes will be made to your existing system unless you install.

🖥️ Option 2: Virtual Machine (VM)

This method runs Linux **inside** your existing OS — no reboot required.

Tools:

- **VirtualBox** (free, cross-platform)
- **VMware Workstation Player** (Windows/Linux)

Steps:

1. Install VirtualBox.
2. Download a Linux ISO (e.g., Ubuntu).
3. Create a new virtual machine in VirtualBox and mount the ISO.
4. Start the VM — you'll boot into Linux like a real PC!

Ideal for experimentation, especially if you don't want to dual-boot.

□ Option 3: Windows Subsystem for Linux (WSL)

If you use Windows 10 or 11, **WSL** lets you run Linux directly inside Windows — no dual boot, no VM.

To set up WSL:

```
wsl --install
```

Then install a distro (Ubuntu is recommended).
WSL gives you a terminal-based Linux environment but **no graphical interface**.

2.2 Installing Linux on Your Computer

When you're ready to make Linux your primary or secondary OS, you can install it directly on your hardware.

📌 Important:

- **Back up your files.** Installation may involve partitioning or formatting your disk.
- Have at least **16GB free space** (32GB recommended).
- Use a **Live USB** to boot the installer.

Installation Process (Ubuntu Example):

1. Boot from your USB stick.
2. Select “Install Ubuntu”.
3. Choose:
    - Install **alongside** Windows (dual-boot)
    - Or **erase disk and install Linux only**
4. Set up:
    - Time zone, keyboard layout
    - Username and password
5. Wait for installation to finish (~15–30 mins), then reboot.

2.3 Understanding Linux Filesystem Layout

Once installed, Linux’s filesystem might look unfamiliar. Here's a cheat sheet:

Directory	Description
`/`	Root directory — top of the tree
`/home`	User folders (`/home/alex`, etc.)
`/etc`	System configuration files
`/bin`	Essential binaries (e.g., `ls`, `cp`)
`/usr`	User apps and libraries
`/var`	Variable files (logs, mail, cache)

Directory	Description
`/tmp`	Temporary files
`/root`	Root user's home directory
`/dev`	Device files
`/proc`	Process information (virtual)

Fun fact: Linux doesn't use drive letters like `C:` — everything is part of one unified tree starting at `/`.

2.4 Choosing Your First Distro (If You Haven't Yet)

If you're unsure which Linux distribution to start with, here are beginner-friendly choices:

Distro	Features
Ubuntu	Polished, popular, great support
Linux Mint	Very Windows-like, easy transition
Zorin OS	Designed for Windows switchers
Pop!_OS	Sleek, great for developers and gamers

All of these work well with most hardware and offer GUI-based installers.

Summary

- You can try Linux using a Live USB, Virtual Machine, or WSL without affecting your current system.
- Installing Linux is straightforward with modern installers.
- The Linux filesystem is structured and logical, though different from Windows.
- Picking the right beginner-friendly distro helps ease your learning curve.

□ Exercises

1. Download a Linux ISO (Ubuntu or Mint). Try booting into it via USB or VirtualBox.
2. Draw a basic Linux directory tree (`/`, `/home`, `/etc`, etc.) and label what each part is for.
3. Research WSL: What are its limitations compared to a full Linux install?

🕮 Chapter 3: Getting Comfortable with the Terminal

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand what the terminal is and why it’s essential in Linux.
- Learn how to open and use the terminal.
- Get familiar with shell basics, common commands, and navigation.
- Start building comfort using the CLI (command-line interface).

3.1 What is the Terminal?

The **terminal** (or command line) is a text-based interface to interact with your Linux system. Instead of clicking buttons, you **type commands** to tell the system what to do.

It might seem intimidating at first — but it's **faster, more powerful**, and gives you **greater control** than any graphical interface.

3.2 Terminal, Shell, and Bash — What’s the Difference?

Term	Description
Terminal	The window you type commands into
Shell	The program that runs your commands (e.g., `bash`, `zsh`)
Bash	The most common shell in Linux (Bourne Again SHell)

When you open a terminal, you're interacting with a shell (usually `bash`), which interprets your commands.

3.3 Opening the Terminal

Depending on your distro, here are ways to open a terminal:

- Press `Ctrl + Alt + T` (most distros)
- Search for "Terminal" in the application menu
- Right-click on the desktop or file manager and choose “Open Terminal”

Once open, you'll see a **prompt** like this:

```
alex@ubuntu:~$
```

This means:

- You're logged in as **alex**
- On the **ubuntu** machine
- In your **home directory** (~)

3.4 Basic Command Structure

The structure of a command is:

```
command [options] [arguments]
```

Example:

```
ls -l /home
```

- `ls` → command
- `-l` → option (long listing format)
- `/home` → argument (the directory to list)

3.5 First Commands to Know

Let's try some essential commands:

Command	Description
`pwd`	Print current directory
`ls`	List files and folders
`cd`	Change directory
`clear`	Clear the screen
`echo`	Print text to terminal
`whoami`	Show your username
`date`	Show current date and time
`man`	Show manual page for a command (e.g., `man ls`)

❗ Try each command now in your terminal!

3.6 Navigating the Filesystem

- `cd /` → go to root directory
- `cd ~` → go to your home directory
- `cd ..` → move up one directory
- `ls` → list contents of current directory

- `ls -a` → list **all** files (including hidden files)
- `ls -lh` → list files with sizes in **human-readable** format

Try this sequence:

```
cd /
ls
cd home
ls
cd ~
```

3.7 Autocomplete and History

- **Autocomplete:** Press `Tab` to autocomplete a file or command.
- **Command history:**
  - Use ↑ and ↓ to scroll through previous commands.
  - `history` shows a list of past commands.
  - `!n` re-runs command #n from the history.

Example:

```
history
!15   # re-runs the 15th command
```

3.8 Tips for Beginners

- Use `man <command>` to read the manual. Example: `man pwd`
- Use `--help` to see quick info. Example: `ls --help`
- Use `Ctrl + C` to **cancel** a running command.
- Practice! The more you type, the faster you'll learn.

Summary

- The terminal gives you direct control of your system.
- You interact using a shell (typically `bash`).
- Commands follow the structure: `command [options] [arguments]`.
- Learning basic navigation (`cd`, `ls`, `pwd`) is key to using Linux comfortably.
- Autocomplete, history, and manual pages are your best friends

□ Exercises

1. Open the terminal and run:
2. `whoami`
3. `pwd`
4. `ls -lh`
5. `date`
6. Navigate from your home directory to `/usr` and back using `cd`.
7. Use `man ls` and write down three useful options for the `ls` command.

🕮 Chapter 4: Navigation and File Management

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand how to navigate the Linux filesystem confidently.
- Learn to create, move, copy, rename, and delete files and directories.
- Grasp the difference between relative and absolute paths.
- Use useful commands to visualize directory structures.

4.1 Navigating the Filesystem

Linux uses a **single hierarchical tree** starting at `/` (root). Knowing how to move around is critical.

Absolute vs Relative Paths

Path Type	Description	Example
Absolute	Full path from root (`/`)	`/home/alex/Documents`
Relative	Path relative to current directory	`Documents` or `../Downloads`

Commands to Know:

Command	Purpose	Example
`pwd`	Show current directory	`pwd`
`ls`	List directory contents	`ls -l`
`cd`	Change directory	`cd /etc` or `cd ..`
`tree`	Display directory tree (may need install)	`tree ~`

4.2 Managing Files and Directories

Creating

Command	Description	Example
`mkdir`	Make new directory	`mkdir projects`
`touch`	Create a new empty file	`touch notes.txt`

Viewing

Command	Description	Example
`cat`	Display file contents	`cat notes.txt`
`head`	Show first lines of a file	`head -n 5 notes.txt`
`tail`	Show last lines of a file	`tail -n 5 notes.txt`
`less`	View file page by page	`less notes.txt`

4.3 Copying, Moving, and Renaming

Command	Description	Example
`cp`	Copy files or directories	`cp notes.txt backup.txt`
`cp -r`	Copy directories recursively	`cp -r projects projects_backup`
`mv`	Move or rename files/directories	`mv notes.txt docs/` or `mv oldname.txt newname.txt`

4.4 Deleting Files and Directories

Be careful! Deletions are permanent in terminal.

Command	Description	Example
`rm`	Remove files	`rm notes.txt`
`rm -r`	Remove directories recursively	`rm -r projects_old`
`rmdir`	Remove empty directories	`rmdir emptyfolder`

4.5 Useful Tips for Managing Files

- Use wildcards `*` to match multiple files.
  Example: `rm *.txt` removes all `.txt` files in the directory.
- Use `ls -lh` for readable file sizes.
- Hidden files start with a dot `.`, use `ls -a` to see them.
- Tab completion helps speed typing filenames.

4.6 Visualizing Directory Structures with `tree`

The `tree` command shows files and folders in a tree-like view.

If `tree` isn't installed:

```
sudo apt install tree  # Debian/Ubuntu
sudo dnf install tree  # Fedora
```

Example:

```
tree ~
```

Summary

- Navigate using `cd` and know when to use absolute or relative paths.
- Create files and directories with `touch` and `mkdir`.
- Copy, move, and rename files with `cp` and `mv`.
- Delete files and directories carefully with `rm` and `rmdir`.
- Use wildcards and tab completion to work faster.
- Visualize structures with `tree`.

Exercises

1. Create a directory called `myproject` in your home folder.
2. Inside `myproject`, create two files: `todo.txt` and `notes.txt`.
3. Copy `todo.txt` to `todo_backup.txt`.
4. Rename `notes.txt` to `meeting_notes.txt`.
5. Delete the backup file.
6. Use `tree` to display your home directory structure.

🕮 Chapter 5: File Operations

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand how to read, create, and edit files.
- Learn commands to view file contents quickly.
- Practice basic text editing in the terminal.
- Get familiar with file manipulation tools.

5.1 Viewing File Contents

Linux offers several commands to view files directly in the terminal:

Command	Description	Example
`cat`	Display the entire file content	`cat notes.txt`
`head`	Show first N lines (default 10)	`head -n 5 notes.txt`
`tail`	Show last N lines (default 10)	`tail -n 5 notes.txt`
`less`	View file one page at a time	`less notes.txt`

`less` is especially useful for large files — navigate with arrow keys, press `q` to quit.

5.2 Creating and Editing Files

Creating Files

- Use `touch filename` to create an empty file.
- Use redirection to write text:

```
echo "Hello Linux!" > hello.txt
```

This creates `hello.txt` with the text inside.

Editing Files with Nano Editor

Nano is a beginner-friendly terminal text editor pre-installed on most distros.

Opening a file:

```
nano notes.txt
```

Basic nano shortcuts:

Shortcut	Action
`Ctrl + O`	Save (Write Out)
`Ctrl + X`	Exit Nano
`Ctrl + K`	Cut a line
`Ctrl + U`	Paste a line
`Ctrl + W`	Search text

Try opening, editing, saving, and exiting a file using nano.

5.3 Copying and Moving Files (Recap)

- Copy file: `cp source.txt destination.txt`
- Move or rename file: `mv oldname.txt newname.txt`

Remember: Use `-r` option with directories.

5.4 Deleting Files and Directories (Recap)

- Delete file: `rm filename`
- Delete directory and contents: `rm -r directoryname`
- Remove empty directory: `rmdir directoryname`

5.5 File Permissions Basics (Preview for Next Chapter)

Files have permissions controlling who can read, write, or execute. We'll dive deep soon, but here's a sneak peek:

Use `ls -l` to see permissions:

```
-rw-r--r-- 1 user user 4096 Jul 15 10:00 notes.txt
```

- `r` = read
- `w` = write
- `x` = execute

Summary

- View files with `cat`, `head`, `tail`, and `less`.
- Create files with `touch` or redirect output with `echo`.
- Edit files easily with the `nano` editor.
- Use `cp`, `mv`, and `rm` to manage files.
- Permissions control access — coming up next!

□ Exercises

1. Create a file called `diary.txt` using `touch`.
2. Open it in `nano`, write a few lines, save, and exit.
3. Use `head` and `tail` to view parts of your file.
4. Copy `diary.txt` to `diary_backup.txt`.
5. Rename the backup file to `diary_old.txt`.
6. Delete `diary_old.txt`.

🕮 Chapter 6: Permissions and Ownership

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand Linux file permission structure.
- Learn how to read permissions using `ls -l`.
- Change permissions with `chmod`.
- Change file ownership with `chown`.
- Get familiar with `sudo` and the role of the root user.

6.1 Why Permissions Matter

Linux is a **multi-user operating system**. File permissions help protect:

- Your personal data
- System files from accidental damage
- Security from unauthorized access

6.2 Understanding `ls -l`

Run:

```
ls -l
```

You'll see something like this:

```
-rw-r--r-- 1 alex alex 1024 Jul 15 10:00 notes.txt
```

Let's break it down:

Part	Meaning
`-rw-r--r--`	File permissions
`1`	Number of hard links
`alex`	Owner (user)
`alex`	Group
`1024`	File size (bytes)
`Jul 15 10:00`	Last modified date/time
`notes.txt`	File name

Permission Breakdown

The first column (e.g., `-rw-r--r--`) has **10 characters**:

```
- rw- r-- r--
|  |   |   |
|  |   |   └── Others
|  |   └────── Group
|  └──────────── Owner
└── File type (- = file, d = directory)
```

Symbol	Meaning
r	read
w	write
x	execute
-	no permission

6.3 Changing Permissions with `chmod`

`chmod` (change mode) modifies who can read, write, or execute a file.

Symbolic Mode

```
chmod u+x script.sh   # Give execute permission to user (owner)
chmod g-w file.txt    # Remove write permission from group
chmod o+r file.txt    # Add read for others
```

Symbol	Refers to
u	user (owner)
g	group
o	others
a	all (u+g+o)

Numeric (Octal) Mode

Permissions can also be represented with **numbers**:

Permission	Value
r	4
w	2
x	1

Example:

```
chmod 755 script.sh
```

Breakdown:

- `7` → user: read (4) + write (2) + execute (1)
- `5` → group: read + execute
- `5` → others: read + execute

More examples:

Command	Description
`chmod 644 file.txt`	rw-r--r-- (standard text file)
`chmod 700 secret.sh`	rwx------ (only owner can access)
`chmod 777 file.txt`	rwxrwxrwx (full access — not safe!)

6.4 Changing Ownership with `chown`

Use `chown` to change file **owner** or **group**:

```
sudo chown bob:bob file.txt
```

- Changes both owner and group to `bob`
- Requires `sudo` (admin rights)

6.5 The Root User and `sudo`

- The **root** user is the superuser with full system access.
- Normal users **don't** have root privileges by default.
- Use `sudo` to run commands **as root**:

```
sudo apt update
sudo rm -rf /some/folder
```

Be careful — root access can **break your system** if misused!

6.6 Viewing Permissions with `stat`

You can use the `stat` command to view detailed metadata:

```
stat file.txt
```

This shows:

- Size
- Access/modification times
- Owner/group
- Permissions in both symbolic and octal format

Summary

- File permissions define what users and groups can do with files.
- Use `ls -l` to view permissions.
- Modify permissions with `chmod`, using symbolic or numeric mode.
- Change ownership with `chown`.
- Use `sudo` carefully for admin-level tasks.

□ Exercises

1. Run `ls -l` in your home directory. Pick a file and explain the permissions.
2. Create a script file (`hello.sh`) and give it execute permission:

```
3. echo 'echo Hello!' > hello.sh
4. chmod +x hello.sh
5. ./hello.sh
```

6. Change the permissions of a file to be:
    - Read-only for everyone: `chmod 444 file.txt`
    - Executable only for owner: `chmod 700 script.sh`
7. Use `stat` on a file and interpret the output.

🕮 Chapter 7: Finding and Searching Files

🎯 Chapter Goals:

By the end of this chapter, you will:

- Locate files and directories using `find`, `locate`, and `which`.
- Search inside files with `grep`.
- Understand wildcards and pattern matching.
- Combine commands for powerful searches.

7.1 Searching with `find`

`find` is a versatile command to search for files **based on name, type, size, date**, and more.

Basic usage:

```
find [starting_point] [condition] [action]
```

Examples:

Command	Description
`find . -name "*.txt"`	Find all `.txt` files in current directory and subdirectories
`find /etc -type d`	Find all directories inside `/etc`
`find / -name passwd 2>/dev/null`	Find files named `passwd` (suppress errors)
`find . -size +1M`	Find files larger than 1MB in current directory

Useful options:

Option	What it does
`-name`	Match by filename
`-type f/d`	Look for files (`f`) or dirs (`d`)
`-size`	Find by file size
`-mtime`	Modified in last N days
`-exec`	Run a command on found files

Example using `-exec`:

```
find . -name "*.log" -exec rm {} \;
```

Deletes all `.log` files — use with caution!

7.2 Using `locate` for Fast Searching

`locate` is much faster than `find` — it uses a pre-built database.

To use it:

```
sudo updatedb     # update the database (usually automatic)
locate bashrc     # find all files containing "bashrc"
```

It shows full paths quickly — great for general searching.

7.3 Using `which` to Locate Executables

If you want to find the location of a **program** (like `ls`, `python`, `nano`):

```
which ls
```

Example output:

```
/bin/ls
```

7.4 Searching Inside Files with `grep`

`grep` searches **within** files for matching text patterns.

Basic usage:

```
grep "pattern" file.txt
```

Examples:

Command	Description
`grep "error" logfile.txt`	Find all lines with the word "error"
`grep -i "linux" file.txt`	Case-insensitive search
`grep -r "TODO" .`	Recursively search in all files in current dir
`grep -n "root" /etc/passwd`	Show line numbers with matches

Combine with `find`:

```
find . -name "*.txt" -exec grep "keyword" {} \;
```

This finds all `.txt` files and searches for a keyword inside them.

7.5 Wildcards and Globbing

Wildcards help match **patterns** when listing or searching for files.

Symbol	Meaning
`*`	Matches any number of characters
`?`	Matches a single character
`[]`	Matches any character in the brackets

Examples:

```
ls *.txt          # all .txt files
ls file?.sh       # file1.sh, fileA.sh
ls log[0-9].txt   # log0.txt to log9.txt
```

7.6 Practice: Combining Tools

You can combine `find`, `grep`, and wildcards for advanced searches.

Example: Find all `.conf` files in `/etc` containing "network":

```
sudo find /etc -name "*.conf" -exec grep -i "network" {} \;
```

Summary

- `find` searches files by name, type, size, date, and more.
- `locate` is faster but relies on a database (`updatedb`).
- `which` tells you where an executable lives.
- `grep` searches inside files for matching text.
- Wildcards (`*`, `?`, `[]`) help match file patterns.

□ Exercises

1. Use `find` to locate all `.sh` files in your home directory.
2. Use `grep` to search for your username in `/etc/passwd`.
3. Try `locate .bashrc` and compare it to `find ~ -name ".bashrc"`.
4. Combine `find` and `grep` to search `.txt` files for the word "backup".
5. Use `ls` with wildcards to list:
    - All files ending in `.log`
    - All files starting with “a” and 3 characters long

🕮 Chapter 8: Installing and Managing Software

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand package managers and their roles.
- Use `apt`, `dnf`, or `pacman` to install, update, and remove software.
- Learn about `.deb`, `.rpm`, and binary packages.
- Discover how to install software from third-party sources and Flatpak/Snap.

8.1 What is a Package Manager?

A **package manager** handles:

- Finding software
- Resolving dependencies
- Installing, upgrading, and removing programs

It’s your one-stop tool for software management.

Distro	Package Manager	File Type
Ubuntu/Debian	`apt` / `dpkg`	`.deb`
Fedora/Red Hat	`dnf` / `rpm`	`.rpm`
Arch Linux	`pacman`	`.pkg.tar.zst`

8.2 Using `apt` (for Ubuntu/Debian)

If you're on Ubuntu, Linux Mint, or Debian, you’ll use `apt`.

Update package lists:

```
sudo apt update
```

Upgrade installed packages:

```
sudo apt upgrade
```

Install software:

```
sudo apt install <package-name>
```

Example:

```
sudo apt install vim
```

Remove software:

```
sudo apt remove <package-name>        # Removes package
sudo apt purge <package-name>         # Removes package + config files
```

Search for packages:

```
apt search <keyword>
```

8.3 Using `dnf` (Fedora/RHEL)

If you're on Fedora or RHEL, the package manager is `dnf`.

Basic commands:

```
sudo dnf install <package>
sudo dnf remove <package>
sudo dnf update
```

You can also search:

```
dnf search <term>
```

8.4 Using `pacman` (Arch Linux & Manjaro)

Arch-based systems use `pacman`.

Common commands:

```
sudo pacman -Syu           # Sync and update
sudo pacman -S <package>   # Install package
sudo pacman -R <package>   # Remove package
```

8.5 Installing `.deb` and `.rpm` Files

Sometimes you may download `.deb` or `.rpm` installer files manually.

On Ubuntu (with `.deb`):

```
sudo dpkg -i package.deb
sudo apt -f install   # fix any missing dependencies
```

On Fedora (with `.rpm`):

```
sudo rpm -i package.rpm
sudo dnf install <package>  # better: handles dependencies
```

8.6 Flatpak and Snap: Universal Packages

These systems work **across distros** and come with sandboxing.

Install Flatpak (if not installed):

```
sudo apt install flatpak
```

Find and install a Flatpak app:

```
flatpak install flathub org.gimp.GIMP
```

Run a Flatpak app:

```
flatpak run org.gimp.GIMP
```

You'll need to add the Flathub repository if it's not already set up.

Snap (used in Ubuntu)

```
sudo snap install vlc
snap list            # list installed snaps
sudo snap remove vlc
```

8.7 Installing from Source (Advanced)

Sometimes software is only available as **source code**.

General steps:

```
sudo apt install build-essential
tar -xvf software.tar.gz
cd software/
./configure
make
sudo make install
```

🛠 Not recommended for beginners unless absolutely necessary.

Summary

- Use your distro's package manager (`apt`, `dnf`, or `pacman`) for most tasks.
- Update and upgrade regularly for security.
- You can manually install `.deb` or `.rpm` files when needed.
- Flatpak and Snap offer universal installation options.
- Source code installation is possible but more complex.

□ Exercises

1. Use your package manager to install:
    - `htop`
    - `curl`
2. Search for an editor (e.g., `vim`, `nano`, or `gedit`) and install it.
3. Remove a program you installed using `apt`, `dnf`, or `pacman`.
4. Try installing and running a Flatpak app from flathub.org.
5. Use `apt list --installed` (or equivalent) to view all installed packages.

🕮 Chapter 9: Users, Groups, and Permissions (Advanced)

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand how Linux handles users and groups.
- Create, modify, and delete users and groups.
- Change file ownership and group access.
- Use `sudo` safely and effectively.
- Understand special permissions: `setuid`, `setgid`, and the `sticky bit`.

9.1 Users and Groups in Linux

Every Linux system has:

- **Users**: individual accounts (e.g., `alex`, `root`, `guest`)
- **Groups**: collections of users that share permissions (e.g., `sudo`, `developers`)

Every file belongs to a **user** and a **group**.

9.2 Viewing Users and Groups

- List all users (from `/etc/passwd`):

```
cat /etc/passwd
```

- List all groups (from `/etc/group`):

```
cat /etc/group
```

- See your current user and groups:

```
whoami
groups
```

9.3 Managing Users

⬢ Requires `sudo` privileges.

Create a new user:

```
sudo adduser john
```

Delete a user:

```
sudo deluser john
```

Change a user's password:

```
sudo passwd john
```

9.4 Managing Groups

Create a new group:

```
sudo groupadd developers
```

Add user to a group:

```
sudo usermod -aG developers john
```

Use `-aG` to **append** (not overwrite) groups.

Remove user from a group (manually edit):

```
sudo gpasswd -d john developers
```

9.5 File Ownership and Permissions (Review)

Use `chown` to change the **owner and group** of a file:

```
sudo chown john:developers project.txt
```

Use `chmod` to manage permissions, e.g.,:

```
chmod 770 shared.txt   # Full access for owner + group
```

9.6 Special Permissions

□ Setuid (`s` on owner execute bit)

If set on a binary, users run it with the **file owner's privileges**.

```
chmod u+s somebinary
```

- Example: `/usr/bin/passwd` allows normal users to change passwords (it runs as root).

□ Setgid (`s` on group execute bit)

- On **files**: program runs with the file's group privileges.
- On **directories**: files created inside inherit the directory's group.

```
chmod g+s shared_dir
```

□ Sticky Bit (`t` on others execute bit)

Used on directories like `/tmp` to let users **only delete their own files**, even if the directory is world-writable.

```
chmod +t /shared_folder
```

Check permissions using:

```
ls -ld /tmp
```

You'll see:

```
drwxrwxrwt ...
```

That final `t` means the sticky bit is set.

9.7 Using `sudo` and `/etc/sudoers`

Give user `sudo` access:

Add them to the `sudo` group:

```
sudo usermod -aG sudo john
```

Editing `sudo` config:

Run:

```
sudo visudo
```

This opens `/etc/sudoers` safely for editing.

Example line:

```
john ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart apache2
```

Allows `john` to restart Apache without a password.

Summary

- Linux uses users and groups to manage permissions.
- Use `adduser`, `deluser`, and `groupadd` to manage accounts.
- Change ownership with `chown`, and permissions with `chmod`.
- Understand and apply special permissions: `setuid`, `setgid`, sticky bit.
- `sudo` gives users temporary admin powers — use wisely.

□ Exercises

1. Create a user called `testuser`.
2. Create a group called `team`, and add `testuser` to it.
3. Create a file, change its owner to `testuser:team`.
4. Set the `setgid` bit on a directory and test file group inheritance.
5. Observe the sticky bit in `/tmp` using `ls -ld /tmp`.

🕮 Chapter 10: Shell Scripting Basics

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand what shell scripts are and how to create them.
- Write and run your first shell script.
- Use variables, input/output, and conditionals.
- Automate simple tasks with loops and commands.
- Set executable permissions and debug basic issues.

10.1 What is a Shell Script?

A **shell script** is a text file containing a series of **commands** written for the shell (usually Bash). It allows you to:

- Automate repetitive tasks
- Chain multiple commands together
- Make logic-based decisions (if/else)
- Process files, logs, and user input

10.2 Your First Shell Script

1. Open a terminal and create a file:

```
nano hello.sh
```

2. Add the following lines:

```
#!/bin/bash
echo "Hello, Linux user!"
```

3. Save and exit (in Nano: `Ctrl + O`, `Enter`, `Ctrl + X`)
4. Make it executable:

```
chmod +x hello.sh
```

5. Run the script:

```
./hello.sh
```

10.3 Script Structure

```
#!/bin/bash      # shebang – tells system to use Bash
# A comment
echo "This is a shell script"
```

Use `#` for comments — helpful for documenting code.

10.4 Variables

Define and use variables like this:

```
name="Alex"
echo "Hello, $name!"
```

- No spaces around `=`
- Use `$variable` to access the value

10.5 Taking User Input

Use `read` to prompt the user:

```
#!/bin/bash
echo "What is your name?"
read name
echo "Nice to meet you, $name!"
```

10.6 If Statements

Basic conditionals:

```
#!/bin/bash
echo "Enter a number:"
read num

if [ "$num" -gt 10 ]; then
    echo "That's greater than 10."
else
    echo "10 or less."
fi
```

Operator	Meaning
`-eq`	equal
`-ne`	not equal
`-lt`	less than
`-le`	less or equal
`-gt`	greater than
`-ge`	greater or equal

10.7 Loops

`for` loop:

```
for i in 1 2 3; do
    echo "Number $i"
done
```

`while` loop:

```
count=1
while [ $count -le 5 ]; do
    echo "Count is $count"
    count=$((count + 1))
done
```

10.8 Running External Commands

Shell scripts can run **any Linux command**:

```
#!/bin/bash
echo "Disk usage:"
df -h

echo "Top 5 processes:"
ps aux | sort -rk 3,3 | head -5
```

10.9 Script Permissions & Execution

Make sure your script is **executable**:

```
chmod +x script.sh
./script.sh
```

Or run it via interpreter:

```
bash script.sh
```

10.10 Debugging Scripts

Use `bash -x script.sh` to see each line as it's executed.

Or insert:

```
set -x   # turn on debug mode
set +x   # turn off debug mode
```

Summary

- Shell scripts are files with a list of Bash commands.
- Use variables, user input, conditionals, and loops to build logic.
- Scripts must be executable (`chmod +x`) to run directly.
- Debugging tools like `bash -x` help troubleshoot issues.

□ Exercises

1. Write a script that:
    - Greets the user by name
    - Asks their age
    - Prints a message based on age (e.g., under or over 18)
2. Create a script that:
    - Lists files in the current directory
    - Asks which file to view
    - Displays that file with `less`
3. Write a `for` loop that:
    - Prints numbers from 1 to 5
    - Creates 5 text files: `file1.txt`, `file2.txt`, etc.

🕮 Chapter 11: Networking Basics in Linux

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand key networking concepts (IP, DNS, routing).
- Use CLI tools to view and manage network interfaces.
- Test connectivity with `ping`, `traceroute`, `netstat`, and more.
- Learn how name resolution works via `/etc/hosts` and DNS.
- Inspect open ports and active connections.
- Troubleshoot common networking issues.
- Understand basic server-client networking setups.

11.1 Key Networking Concepts

Before jumping into commands, let's quickly review some essential concepts:

Term	Description
IP Address	Unique address of a device on a network (e.g., `192.168.1.10`)
Subnet Mask	Defines network vs host portion of IP (e.g., `255.255.255.0`)
Gateway	Router that connects your system to other networks
DNS	Resolves domain names (like `google.com`) into IP addresses
MAC Address	Unique hardware identifier of network interface
Port	Software endpoint on a machine (e.g., SSH = port 22, HTTP = port 80)

11.2 Viewing Network Interfaces

Use the `ip` command (replaces older `ifconfig`):

```
ip addr          # Show all interfaces and IPs
ip link          # Show interfaces and their status
ip route         # Show routing table
```

Sample output of `ip addr`:

```
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
    inet 192.168.1.100/24 brd 192.168.1.255 scope global dynamic enp0s3
```

- `inet`: IPv4 address
- `enp0s3`: Interface name

To see just your IP:

```
hostname -I
```

11.3 Testing Connectivity

`ping`

Sends ICMP echo requests to test if a host is reachable.

```
ping google.com
ping 8.8.8.8
```

Use `Ctrl + C` to stop it.

`traceroute`

Shows the path packets take to reach a destination.

```
traceroute google.com
```

(You might need to install it first: `sudo apt install traceroute`)

`curl` and `wget`

Used to retrieve data from URLs — useful for testing HTTP:

```
curl https://example.com
wget https://example.com
```

11.4 Managing Interfaces (Advanced Use)

Bring interface up/down:

```
sudo ip link set enp0s3 down
sudo ip link set enp0s3 up
```

Assign static IP temporarily:

```
sudo ip addr add 192.168.1.50/24 dev enp0s3
```

This change lasts until reboot. For permanent configs, edit `/etc/network/interfaces` (Debian) or use `nmcli` or `netplan`.

11.5 DNS and Hostname Resolution

Check DNS info:

```
cat /etc/resolv.conf
```

This file typically shows:

```
nameserver 8.8.8.8
```

Check your hostname:

```
hostname
```

Change it temporarily:

```
sudo hostname newname
```

`/etc/hosts` file

Manually maps names to IPs. Useful for testing:

```
127.0.0.1   localhost
192.168.1.20   testserver.local
```

11.6 Viewing Open Ports and Services

`ss` — socket statistics

```
ss -tuln
```

- `-t`: TCP
- `-u`: UDP
- `-l`: listening
- `-n`: numeric (don’t resolve names)

You’ll see open ports and what services are listening.

`netstat` (older alternative):

```
netstat -tuln
```

(Install it via `sudo apt install net-tools`)

11.7 Network Service Management

To check if a service is reachable:

```
telnet example.com 80
```

Or use `nc` (netcat):

```
nc -zv example.com 22   # test if SSH is open
```

Install netcat if needed:

```
sudo apt install netcat
```

11.8 Common Services and Ports

Service	Port
HTTP	80
HTTPS	443
SSH	22
FTP	21
DNS	53
SMTP	25

11.9 Troubleshooting Network Issues

Problem	Try This
No internet	`ping 8.8.8.8` or check DNS (`resolv.conf`)
Can't connect to a host	`ping`, `traceroute`, or `nc`
Service not reachable	Check firewall or `ss -tuln`
Interface down	`ip link set` or `nmcli`
DNS issues	Try pinging IP directly or check `/etc/resolv.conf`

11.10 Bonus: Simple Network Tools

- `nmcli` — CLI for managing NetworkManager (used in Ubuntu GUI)
- `nmtui` — TUI for network settings
- `iftop` — real-time bandwidth monitor (`sudo apt install iftop`)
- `nmap` — scan open ports on other systems

Example:

```
nmap -p 22,80,443 192.168.1.1
```

Summary

- Use `ip`, `ping`, `traceroute`, and `ss` to manage and debug networking.
- Understand IPs, gateways, DNS, and routing basics.
- Monitor open ports and active services with `ss` or `netstat`.
- Manually control name resolution via `/etc/hosts`.
- Tools like `nmap`, `netcat`, and `nmcli` enhance your network visibility.

□ Exercises

1. Find your IP address using `ip addr`.
2. Use `ping` and `traceroute` to check connectivity to a website.
3. List all open ports on your system with `ss -tuln`.
4. Add an entry to `/etc/hosts` and try pinging it.
5. Use `nc` or `nmap` to check if a local device has SSH enabled.
6. Temporarily assign a static IP to your network interface.
7. Check and interpret the output of `ip route`.

🕮 Chapter 12: File Archiving, Compression, and Backups

🎯 Chapter Goals:

By the end of this chapter, you will:

- Create and extract archive files using `tar`.
- Compress and decompress files using `gzip`, `bzip2`, and `xz`.
- Use `zip` and `unzip` for compatibility with other systems.
- Understand the structure of backup strategies.
- Automate backups with shell scripts and `cron`.

12.1 Archiving with `tar`

The `tar` command combines multiple files into a single **archive file** (often called a *tarball*), with or without compression.

Create a tar archive:

```
tar -cvf archive.tar folder/
```

- `c`: create
- `v`: verbose (show progress)
- `f`: file name

Extract a tar archive:

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

List contents of a tar archive:

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

12.2 Adding Compression

You can compress the tarball in one step:

Compression	Command	File Extension
gzip	`tar -czvf file.tar.gz`	`.tar.gz` or `.tgz`
bzip2	`tar -cjvf file.tar.bz2`	`.tar.bz2`
xz	`tar -cJvf file.tar.xz`	`.tar.xz`

Extract compressed tar files:

```
tar -xzvf archive.tar.gz    # gzip
tar -xjvf archive.tar.bz2   # bzip2
tar -xJvf archive.tar.xz    # xz
```

12.3 Using `gzip`, `bzip2`, and `xz` Directly

You can also compress **single files** without using `tar`:

```
gzip file.txt        # Creates file.txt.gz
bzip2 file.txt       # Creates file.txt.bz2
xz file.txt          # Creates file.txt.xz
```

To decompress:

```
gunzip file.txt.gz
bunzip2 file.txt.bz2
unxz file.txt.xz
```

12.4 Using `zip` and `unzip`

`zip` is popular for Windows-compatible archives.

```
zip archive.zip file1 file2 folder/
unzip archive.zip
```

To list contents:

```
unzip -l archive.zip
```

You can add files incrementally:

```
zip archive.zip newfile.txt
```

12.5 Splitting Large Archives

Split a large archive into parts (e.g., for USB or email limits):

```
split -b 100M bigfile.tar.gz part_
```

Reassemble:

```
cat part_* > bigfile.tar.gz
```

12.6 Backups: Strategy and Practice

A good backup is **automated**, **tested**, and **stored off-device**.

Backup types:

Type	Description
Full	Everything copied
Incremental	Only changes since last backup
Differential	Changes since last full backup

Example: Simple Manual Backup Script

```
#!/bin/bash

backup_dir="/home/user/backups"
src_dir="/home/user/Documents"
date=$(date +%F)
filename="backup-$date.tar.gz"

tar -czvf "$backup_dir/$filename" "$src_dir"
```

Make it executable:

```
chmod +x backup.sh
```

Run it:

```
./backup.sh
```

12.7 Automating Backups with `cron`

Edit your crontab:

```
crontab -e
```

Run backup script every day at 2AM:

```
0 2 * * * /home/user/backup.sh
```

Learn more in Chapter 14 on scheduling with `cron`.

12.8 Checking Disk Usage

Before and after backups, check space usage:

```
df -h         # Disk space by filesystem
du -sh *      # Size of each item in current directory
```

Summary

- `tar` is used for creating archive files; add `gzip`, `bzip2`, or `xz` to compress.
- `zip` and `unzip` work cross-platform.
- Scripts and `cron` make backups automatic.
- Backup strategies include full, incremental, and differential methods.

□ Exercises

1. Create a `.tar.gz` backup of your home directory (or a folder inside it).
2. Extract a `.tar.bz2` file and list its contents.
3. Compress a text file using `gzip`, then decompress it.
4. Write a shell script to back up a folder to `/tmp/backups/`.
5. Set a cron job to run that script daily.
6. Use `du` and `df` to monitor your disk before and after backup.

🕮 Chapter 13: Package Building and Software Compilation

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand what it means to compile software from source.
- Install essential development tools (`build-essential`, `gcc`, `make`, etc.).
- Download, configure, build, and install open-source programs manually.
- Learn how to create a `.deb` or `.rpm` package for easy software distribution.
- Understand what "dependency hell" is — and how package managers help.

13.1 What Does It Mean to Build Software?

When you install software from a package (`apt`, `dnf`, etc.), it's **already compiled** — i.e., turned from human-readable source code into machine code.

But many open-source projects provide only **source code**, which you must compile yourself.

🛠 You'll often see this with niche software, GitHub projects, or bleeding-edge tools.

13.2 Required Tools: Installing a Build Environment

On Debian/Ubuntu systems:

```
sudo apt update
sudo apt install build-essential
```

This installs:

- `gcc`: GNU C Compiler
- `g++`: C++ compiler
- `make`: build system tool
- `libc6-dev`: development headers

For Red Hat/Fedora:

```
sudo dnf groupinstall "Development Tools"
```

13.3 Typical Build Workflow from Source

Open-source software usually comes as a compressed tarball (`.tar.gz` or `.tar.bz2`) or via Git.

1. Download and extract:

```
wget https://example.com/project.tar.gz
tar -xzf project.tar.gz
cd project/
```

2. Configure the build system:

```
./configure
```

This checks for required libraries and prepares `Makefile`.

Options you might pass:

```
./configure --prefix=/opt/myapp --enable-feature-x
```

3. Compile the source:

```
make
```

This step may take time depending on the size of the project.

4. Install the compiled binary:

```
sudo make install
```

This copies files into appropriate locations (like `/usr/local/bin`).

You can uninstall by running `sudo make uninstall` — if supported.

13.4 Problems You Might Encounter

Missing dependencies:

During `./configure`, you might see errors like:

```
configure: error: missing required library xyz
```

Use `apt` or `dnf` to install development packages, usually named like `libxyz-dev`.

Use `apt-cache search` or `dnf search` to find the exact package.

13.5 Building Software from Git

More and more projects are hosted on GitHub or GitLab.

```
git clone https://github.com/example/project.git
cd project
```

Then follow the same steps: often `./configure && make && sudo make install`.

Some use `cmake`, `meson`, or `ninja` instead of `configure/make`.

13.6 Building Your Own `.deb` Package (Debian/Ubuntu)

Packaging your own software makes installation **easier and cleaner**.

Example directory layout:

```
myapp/
├── DEBIAN/
│   └── control
└── usr/
    └── local/
        └── bin/
            └── myapp
```

Sample `control` file:

```
Package: myapp
Version: 1.0
Section: base
Priority: optional
Architecture: amd64
Maintainer: You <you@example.com>
Description: A sample app packaged as .deb
```

Build the package:

```
dpkg-deb --build myapp
```

Result:

```
myapp.deb
```

Install it:

```
sudo dpkg -i myapp.deb
```

Use `dpkg -r myapp` to remove it later.

13.7 Building `.rpm` Packages (Fedora, RHEL)

This process uses `rpmbuild`. Install the tools:

```
sudo dnf install rpm-build
```

Package specs are written in `.spec` files. The process is more involved than `.deb`, but the concepts are the same.

13.8 Managing Your Builds

Where installed files go:

By default, `make install` might place files in:

- `/usr/local/bin`
- `/usr/local/lib`

To track them, you can install using `checkinstall`:

```
sudo apt install checkinstall
sudo checkinstall
```

This turns your manual install into a `.deb` you can uninstall later.

Summary

- Compiling software from source gives flexibility, but requires build tools.
- Use `configure`, `make`, and `make install` for most projects.
- Track and manage installations to avoid cluttering your system.
- You can package software using `.deb` or `.rpm` formats for easy deployment.
- Tools like `checkinstall` and `fpm` make packaging easier.

□ Exercises

1. Install `build-essential` (or your distro's equivalent).
2. Download and compile a small open-source project from source.
3. Try creating a `.deb` package for a basic shell script placed in `/usr/local/bin/`.
4. Uninstall a compiled program using `make uninstall` or `dpkg -r`.
5. Clone a GitHub repo and build it manually.
6. (Bonus) Try using `checkinstall` to build and track a package.

🕮 Chapter 14: Automating Tasks with `cron` and `at`

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand how `cron` schedules recurring jobs.
- Use `crontab` to create and manage scheduled tasks.
- Read and write cron job syntax correctly.
- Understand `at` for one-time tasks.
- View and debug logs for scheduled task execution.

14.1 Why Automate?

Linux thrives on **automation**. Cron jobs can:

- Run backups nightly.
- Send email alerts.
- Rotate logs weekly.
- Sync data every 10 minutes.

Automation removes human error, saves time, and keeps systems stable.

14.2 Understanding `cron`

The `cron` daemon is always running in the background, checking for tasks to execute based on **time-based rules**.

Each user (even root) can define a personal list of cron jobs using a **crontab**.

14.3 Cron Syntax

A cron job has **5 time fields** and then a command:

```
MIN  HOUR  DOM  MON  DOW  command
```

Field		**Meaning**	**Values**
MIN		Minute	0–59
HOUR		Hour	0–23
DOM		Day of Month	1–31
MON		Month	1–12
DOW		Day of Week	0–7 (0 and 7 = Sun)

Examples:

```
0 3 * * * /home/user/backup.sh       # Every day at 3:00 AM
30 2 * * 1 /home/user/script.sh      # Every Monday at 2:30 AM
*/10 * * * * /home/user/check.sh     # Every 10 minutes
```

14.4 Managing Your Crontab

Edit your crontab:

```
crontab -e
```

This opens your personal cron file in the default editor.

View your cron jobs:

```
crontab -l
```

Remove your crontab:

```
crontab -r
```

14.5 Cron Job Output and Logging

By default, output from cron jobs is **emailed to your user account** (if email is configured).

To log output manually:

```
0 2 * * * /home/user/script.sh >> /home/user/logs/cron.log 2>&1
```

- `>>`: append output
- `2>&1`: also capture errors

14.6 System-Wide Cron Jobs

Besides per-user crontabs, there are global files:

- `/etc/crontab`
- `/etc/cron.d/` — drop-in config files
- `/etc/cron.hourly/`, `/daily/`, `/weekly/`, etc.

Format in `/etc/crontab` includes **username**:

```
MIN HOUR DOM MON DOW USER COMMAND
```

14.7 Using `at` for One-Time Tasks

Use `at` when you need to **schedule something once**, like a script reboot or reminder.

Check if `atd` is running:

```
sudo systemctl status atd
```

Start it if needed:

```
sudo systemctl start atd
```

Schedule a one-time command:

```
at now + 2 minutes
```

Then type a command:

```
echo "Reminder!" >> ~/reminders.txt
<Ctrl + D>
```

View scheduled tasks:

```
atq
```

Remove a scheduled task:

```
atrm <job-number>
```

14.8 Environment in Cron

Cron runs in a **limited environment**, so you may need to:

- Use full paths (e.g., `/usr/bin/python3` instead of `python3`)
- Set environment variables inside the script
- Add `PATH` manually in the crontab:

```
PATH=/usr/bin:/bin:/usr/local/bin
```

14.9Best Practices

1. Test your script manually before using cron.
2. Always log output (`>> logfile 2>&1`).
3. Use full paths in commands.
4. Use `crontab -l > backup.txt` to save your job list.
5. Avoid `* * * * *` unless you really need per-minute jobs.

Summary

- `cron` schedules **repeating** tasks using time-based rules.
- `at` schedules **one-time** tasks.
- Use `crontab -e` to create personal cron jobs.
- Log output and errors for debugging.
- `cron` runs with limited environment — use absolute paths.

□ Exercises

1. Use `crontab -e` to schedule a script to run every day at 6 AM.
2. Redirect your script's output to a log file in your home directory.
3. Schedule a one-time job with `at` that writes to a file 5 minutes from now.
4. Use `atq` and `atrm` to manage a job you created.
5. Set up a weekly cron job to archive your `~/Documents` folder.

🕮 Chapter 15: System Logging and Log Analysis

🎯 Chapter Goals:

By the end of this chapter, you will:

- Understand how Linux logs system events.
- Explore important log files under `/var/log/`.
- Use tools like `journalctl`, `dmesg`, and `tail` to read logs.
- Filter and search logs effectively.
- Know where to look for boot issues, service failures, and user activity.

15.1 What Are Logs?

Logs are **system-generated records** that keep track of:

- Kernel and hardware events
- System boots and shutdowns
- Application crashes
- User logins and sudo activity
- Network and security events
- Cron jobs, services, and more

They are essential for:

- Debugging problems
- Auditing user actions
- Security forensics
- Monitoring system health

15.2 Key Log File Locations

Most log files live under:

```
/var/log/
```

Here are some common ones:

File	Purpose
`/var/log/syslog` (Debian)	General system messages
`/var/log/messages` (RHEL)	Similar to syslog

File	Purpose
`/var/log/dmesg`	Kernel ring buffer (hardware/boot)
`/var/log/auth.log`	Logins, sudo, authentication
`/var/log/kern.log`	Kernel messages
`/var/log/boot.log`	Boot process log
`/var/log/cron.log` or `cron`	Cron job logs
`/var/log/Xorg.0.log`	X (GUI) session logs
`/var/log/nginx/` or `/apache2/`	Web server logs

15.3 Reading Log Files

Use basic tools like:

```
less /var/log/syslog
tail -f /var/log/syslog      # Follow new log entries in real time
```

View last N lines:

```
tail -n 50 /var/log/auth.log
```

15.4 `journalctl`: The Systemd Log Viewer

Most modern Linux systems use **systemd**, and `journalctl` reads its logs.

View full journal:

```
journalctl
```

Most recent logs:

```
journalctl -r         # Reverse order (latest first)
journalctl -n 100     # Last 100 lines
```

Logs from current boot only:

```
journalctl -b
```

To see previous boots:

```
journalctl --list-boots
journalctl -b -1       # Last boot
journalctl -b -2       # Two boots ago
```

Logs for a specific service:

```
journalctl -u ssh
journalctl -u nginx.service
```

Logs for a time range:

```
journalctl --since "2024-12-01" --until "2024-12-02"
```

15.5 `dmesg`: Kernel and Hardware Logs

`dmesg` shows messages from the **kernel ring buffer**:

```
dmesg
dmesg | grep usb      # Filter for USB-related events
```

Useful during:

- Boot analysis
- Hardware issues (e.g., USB, disk failures)

15.6 Log Rotation with `logrotate`

Logs grow quickly — `logrotate` rotates (archives) them regularly.

Config files:

- `/etc/logrotate.conf` – global
- `/etc/logrotate.d/` – per-service

A rotated log might look like:

```
/var/log/syslog
/var/log/syslog.1
/var/log/syslog.2.gz
```

Manual rotation:

```
sudo logrotate /etc/logrotate.conf
```

15.7 Searching Logs

Using `grep`:

```
grep ssh /var/log/auth.log
grep -i error /var/log/syslog
```

Combine with `journalctl`:

```
journalctl | grep -i fail
journalctl -u ssh | grep "Accepted"
```

15.8 Monitoring Logs in Real Time

Watch logs live:

```
tail -f /var/log/syslog
journalctl -f
```

This is ideal for monitoring while triggering an action (e.g., running a failing service, plugging in a USB drive, etc.).

Summary

- Linux logs are mainly stored under `/var/log/`.
- Use `journalctl` for systemd logs, and `dmesg` for kernel-level messages.
- Tools like `less`, `tail`, and `grep` help read and filter logs.
- Logs can be rotated automatically with `logrotate`.

□ Exercises

1. View the last 100 lines of `/var/log/auth.log` using `tail` and `less`.
2. Use `journalctl -u ssh` to inspect SSH activity on your system.
3. Filter dmesg logs for USB device activity.
4. Check when your system last booted using `journalctl --list-boots`.
5. Observe log changes live while restarting a service with `journalctl -f`.

🕮 Chapter XX: Linux Keyboard Shortcuts at a Glance

🎯 Chapter Goals:

- Memorize essential keyboard shortcuts for the terminal and shell.
- Learn shortcuts for command line editing, navigation, and job control.
- Speed up command execution with history and autocomplete shortcuts.
- Improve productivity with multitasking and window management keys.

1. Terminal & Shell Editing Shortcuts

Shortcut	Action	Description
`Ctrl + A`	Move cursor to the beginning of the line	Quickly jump to line start
`Ctrl + E`	Move cursor to the end of the line	Jump to line end
`Ctrl + U`	Cut (delete) from cursor to beginning of line	Clear from cursor backward
`Ctrl + K`	Cut (delete) from cursor to end of line	Clear from cursor forward
`Ctrl + W`	Cut (delete) word before cursor	Delete previous word
`Ctrl + Y`	Paste the last cut text	Yank (paste) text back
`Ctrl + D`	Delete character under cursor / Exit shell if line empty	Delete char or exit terminal if empty
`Ctrl + H` or `Backspace`	Delete character before cursor	Backspace delete
`Ctrl + T`	Swap character under cursor with previous one	Fix typos by swapping
`Alt + B`	Move cursor backward one word	Move word left
`Alt + F`	Move cursor forward one word	Move word right

2. Command History Shortcuts

Shortcut	Action	Description
`Ctrl + R`	Reverse search command history	Search previous commands interactively
`Ctrl + G`	Cancel history search	Exit reverse search mode
`Up Arrow`	Scroll backward through command history	Recall previous commands
`Down Arrow`	Scroll forward through command history	Move to newer commands

Shortcut	Action	Description
`!!`	Repeat last command	Quickly rerun last command
`!n`	Run command number `n` from history	Execute a specific history entry

3. Job Control Shortcuts

Shortcut	Action	Description
`Ctrl + C`	Kill/interrupt current running process	Stop current command
`Ctrl + Z`	Suspend current running process	Pause and background current job
`fg`	Resume the last suspended job in the foreground	Bring job back to foreground
`bg`	Resume the last suspended job in the background	Continue job in background
`jobs`	List background and suspended jobs	See job status

4. Autocomplete and Command Completion

Shortcut	Action	Description
`Tab`	Autocomplete commands, file names, variables	Press once or twice for suggestions
`Esc + Esc`	List possible completions if ambiguous	Show list of options

5. Terminal Multiplexer Shortcuts (tmux / screen)

If you use **tmux** or **screen**, these shortcuts are essential:

Shortcut	Action	Description
`Ctrl + B` then `C`	Create a new window	Start a new tmux window
`Ctrl + B` then `N`	Next window	Switch to next tmux window
`Ctrl + B` then `P`	Previous window	Switch to previous tmux window
`Ctrl + B` then `%`	Split window vertically	Split pane vertically
`Ctrl + B` then `"`	Split window horizontally	Split pane horizontally
`Ctrl + B` then `D`	Detach session	Detach tmux session and return to shell

6. Screen Clearing and Terminal Control

Shortcut	Action	Description
`Ctrl + L`	Clear terminal screen	Same as `clear` command
`reset`	Reset the terminal	Fix messed-up terminal display
`Ctrl + S`	Stop terminal output (XOFF)	Pause output if flooding terminal
`Ctrl + Q`	Resume terminal output (XON)	Resume output after Ctrl+S

7. Miscellaneous Shortcuts

Shortcut	Action	Description
`Ctrl + X` then `E`	Open current command line in default editor	Edit command before execution
`Ctrl + V`	Insert next typed character literally	Useful for inserting control characters
`Alt + .`	Insert last argument of previous command	Quickly reuse last argument
`Ctrl + P`	Previous command (similar to Up Arrow)	Recall last command
`Ctrl + N`	Next command (similar to Down Arrow)	Forward in history

Tips to Practice:

- Try navigating and editing long commands with `Ctrl + A`, `Ctrl + E`, `Alt + B`, and `Alt + F`.
- Use `Ctrl + R` to search for previous commands instead of scrolling with arrows.
- Manage jobs actively with `Ctrl + Z`, `bg`, and `fg`.
- Use `Tab` autocomplete everywhere to save typing.
- Remember `Ctrl + L` to quickly clear cluttered terminals.
- Combine shortcuts to increase efficiency, e.g., recall last command with `Ctrl + R` then edit with `Ctrl + A`.