🐧 Linux Mastery Course

by Loki & Shelly

> **Course Focus:** Beginner to Intermediate+ Linux for CyberSecurity
> **Target Audience:** CSIT Students, Aspiring Ethical Hackers, SysAdmins
> **Total Modules:** 17

Module 1: Foundation & Philosophy

🎯 Introduction

Linux is a **Unix-like operating system** built around the Linux kernel. It powers everything from servers to smartphones, and is the **backbone of cybersecurity** operations worldwide. Understanding the **kernel**, **shell**, and **user space** is critical before diving into commands. Without this foundation, you'll struggle with advanced topics.

This module covers what Linux actually is, why it's the go-to OS for security professionals, and how to set up your own hacking lab.

1.1 What is Linux?

Linux is not just an operating system — it's a **kernel** combined with **GNU utilities** to form a complete OS.

Component	Description
Kernel	Core component that manages CPU, memory, devices, and system calls
Shell	Command-line interpreter (bash, zsh, sh) that processes user commands
User Space	Where applications and user processes run, isolated from kernel space
GNU Tools	Essential utilities (grep, sed, awk, tar) that make Linux functional

> 🐧 **GNU/Linux:** The technically correct term — GNU provides the tools, Linux provides the kernel.

1.2 Why Linux for CyberSecurity?

Linux dominates the cybersecurity landscape:

Reason	Explanation
Open Source	Full transparency — audit every line of code
Flexibility	Customize everything from kernel to desktop
Security	Multi-user security model, strong permissions
Tool Ecosystem	Most security tools are Linux-native (Metasploit, Nmap, Burp Suite)
Server Dominance	96%+ of web servers run Linux — mandatory knowledge for pentesting
Scripting Power	Bash scripting enables automation of complex tasks

---

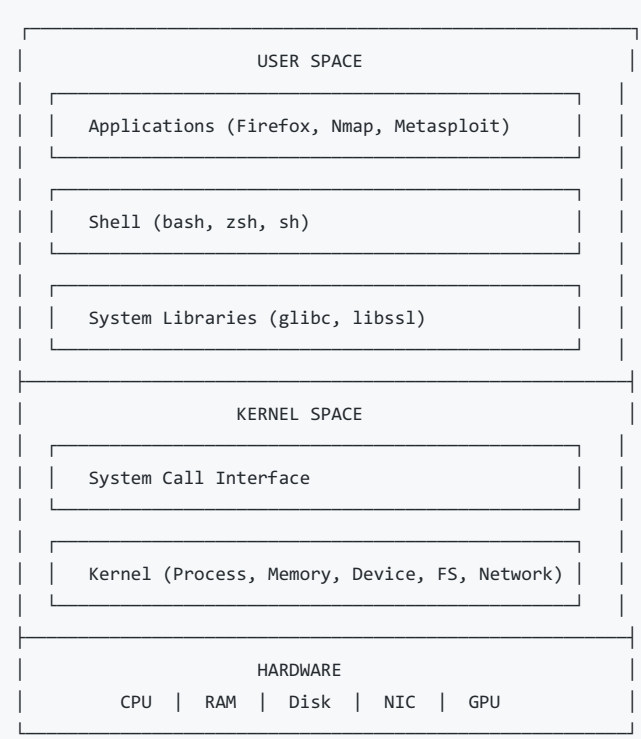
1.3 Linux vs Windows vs macOS

Feature	Linux	Windows	macOS

Feature	Linux	Windows	macOS
Source Code	Open Source	Proprietary	Proprietary (BSD-based kernel)
Cost	Free	Paid License	Included with Apple hardware
Security Tools	Native support	Limited/Ported	Some native, many ported
Customization	Unlimited	Limited	Limited
Server Usage	~96% of servers	~4% of servers	Rare
Package Manager	apt, yum, pacman	None native (winget new)	Homebrew (third-party)
Filesystem	ext4, xfs, btrfs	NTFS, FAT32	APFS, HFS+
Pentesting Distros	Kali, Parrot, BlackArch	None	None

> ⚠ **Warning:** Windows is the target, Linux is the weapon. Know both.

1.4 Architecture Overview



Key Layers:

- **User Space:** Where you (and hackers) operate — running applications and commands
- **Kernel Space:** Protected memory where the OS core runs — handles hardware abstraction
- **Hardware:** Physical components the kernel manages

1.5 Common Linux Distributions

Distro	Base	Package Manager	Best For
Kali Linux	Debian	apt	Penetration Testing, Ethical Hacking
Parrot OS	Debian	apt	Pentesting + Privacy + Development
Ubuntu	Debian	apt	General use, Beginners, Servers

Distro	Base	Package Manager	Best For
Debian	Independent	apt	Stability, Servers
Arch Linux	Independent	pacman	Advanced users, Customization
Fedora	Independent	dnf	Cutting-edge features, Developers
CentOS/RHEL	Independent	yum/dnf	Enterprise servers
BlackArch	Arch	pacman	Advanced Pentesting (2800+ tools)

> **For CyberSec:** Start with **Kali Linux** — 600+ pre-installed security tools.

1.6 Virtual Machine Setup (Best Practices)

Recommended Setup for Your System

> **Your System:** Windows 11 Pro (Acer Nitro V15, RTX 3050, 16GB DDR5) + VirtualBox

VirtualBox Configuration for Kali Linux

Setting	Recommended Value	Reason
RAM	4096 MB (4 GB)	Smooth operation with multiple tools
CPU Cores	2-4 cores	Parallel processing for scans
Storage	80 GB (Dynamic)	Tools + wordlists + captures
Network	NAT + Host-Only	NAT for internet, Host-Only for lab
Graphics	128 MB VRAM	GUI performance
Clipboard	Bidirectional	Copy-paste between host and VM
Drag & Drop	Bidirectional	Easy file transfer

Post-Installation Commands

```
# Update system immediately after install
sudo apt update && sudo apt upgrade -y

# Install VirtualBox Guest Additions (for better performance)
sudo apt install -y virtualbox-guest-x11

# Verify installation
uname -a
lsb_release -a
```

Snapshot Strategy

Snapshot Name	When to Take
`fresh-install`	Right after OS installation
`base-configured`	After updates + guest additions
`pre-engagement`	Before starting any pentest lab

> ⚠ **Warning:** Always snapshot before experiments. Breaking your VM is part of learning — recovering should be instant.

Key Terms

Term	Definition
Kernel	Core of the OS — manages hardware, memory, processes. Direct communication with CPU.
Shell	Command interpreter — translates your commands into kernel actions. Examples: bash, zsh.
Distro	Distribution — a complete OS package built around the Linux kernel (Kali, Ubuntu, Arch).
GNU	"GNU's Not Unix" — collection of free software tools that complement the Linux kernel.
User Space	Memory area where applications run — isolated from kernel for security.
Kernel Space	Protected memory where OS core operates — only kernel code runs here.
Package Manager	Tool for installing/updating software (apt, yum, pacman).
Rolling Release	Continuous updates without major version upgrades (Arch, Kali).
LTS	Long Term Support — stable releases with extended security updates (Ubuntu LTS).

FAQs

Q: Why use Kali Linux instead of Ubuntu for CyberSec?

> Kali comes pre-installed with **600+ penetration testing tools** — Metasploit, Nmap, Burp Suite, John the Ripper, etc. Ubuntu is general-purpose and requires manual installation of each tool. For learning, Kali gets you hacking faster.

Q: Is Kali Linux illegal?

> No. Kali is legal to download and use. **How you use it** determines legality. Scanning/attacking systems without permission is illegal. Always get written authorization.

Q: Should I dual-boot or use a VM?

> **Start with VM. ** Reasons:
>
> - Snapshots save you from breaking things
> - Isolated network for safe lab practice
> - Easy to reset and experiment
> - No risk to your main Windows installation
>
> Dual-boot later when you need bare-metal performance for specific tasks.

Q: What's the difference between bash and zsh?

> Both are shells. **bash** (Bourne Again Shell) is the default on most systems. **zsh** has better autocompletion, themes, and plugins. Kali uses zsh by default since 2020. Core commands work the same in both.

Q: Can I run Linux tools on Windows directly?

> Yes, via **WSL2** (Windows Subsystem for Linux). However, some low-level network tools (raw sockets, packet injection) don't work properly. For serious pentesting, use a full VM or bare-metal.

Practice Tasks

Task 1: Install Kali Linux on VirtualBox

Objective: Set up your hacking lab

Steps:

1. Download Kali Linux VirtualBox image from kali.org
2. Import the `.ova` file into VirtualBox
3. Configure RAM (4GB), CPU (2 cores), Network (NAT)
4. Boot and login (default: `kali / kali`)
5. Change default password: `passwd`
6. Update system: `sudo apt update && sudo apt upgrade -y`

7. Take a snapshot named `base-configured`

Deliverable: Screenshot of `neofetch` or `uname -a` output

Task 2: Identify Your Kernel Version and Architecture

Objective: Understand your system internals

Commands to run:

```
uname -r          # Kernel version
uname -m          # Architecture (x86_64 = 64-bit)
uname -a          # All system info
lsb_release -a    # Distribution details
cat /etc/os-release  # Alternative distro info
```

Deliverable: Document your findings in this format:

```
Kernel Version: _____________
Architecture:  _____________
Distro: _____________
Codename: _____________
```

Task 3: Compare 3 Distros

Objective: Understand the Linux ecosystem

Research and document:

Criteria	Kali Linux	Ubuntu	Arch Linux
Primary Use Case			
Package Manager			
Release Model			
Default Desktop			
Beginner Friendly?			
Pre-installed Security Tools			

Deliverable: Completed comparison table with your findings

✅ Module 1 Checkpoint

Before moving to Module 2, confirm:

- ☐ Kali Linux VM installed and running
- ☐ Default password changed
- ☐ System updated
- ☐ Snapshot taken
- ☐ Kernel version and architecture identified
- ☐ Understood the difference between kernel, shell, and user space

---

Module 2: Getting Started — The Shell

Introduction

The shell is your **primary interface** to Linux. Every command you type, every script you run, every hack you execute — it all goes through the shell. Mastering shell navigation and basic file operations is **non-negotiable** for any security professional.

This module covers terminal fundamentals, navigation commands, and essential file operations. By the end, you'll move through the Linux filesystem like second nature.

2.1 Terminal vs Shell vs Console

These terms are often confused. Here's the clarity:

Term	Definition
Terminal	Application/window that displays the shell (GNOME Terminal, Konsole, xterm)
Shell	The actual command interpreter running inside the terminal (bash, zsh)
Console	Physical/system-level terminal (Ctrl+Alt+F1 on Linux) — no GUI
TTY	Teletypewriter — legacy term for terminal devices (`/dev/tty`)
Prompt	The text indicating the shell is ready for input (`┌──(kali㉿kali)-[~]`)

> **Think of it this way:** Terminal is the TV, Shell is the channel, Console is the antenna.

2.2 Shell Types

Shell	Full Name	Description
bash	Bourne Again Shell	Most common, default on most Linux distros
zsh	Z Shell	Enhanced bash with better autocompletion, Kali default
sh	Bourne Shell	Original Unix shell, minimal, POSIX-compliant
fish	Friendly Interactive Shell	User-friendly, syntax highlighting, not POSIX
dash	Debian Almquist Shell	Lightweight, used for system scripts

Check Your Current Shell

```
echo $SHELL           # Default shell
echo $0               # Current shell
cat /etc/shells       # All available shells
```

Change Default Shell

```
chsh -s /bin/bash     # Switch to bash
chsh -s /bin/zsh      # Switch to zsh
```

> ⚠ **For scripting:** Always use `#!/bin/bash` or `#!/bin/sh` for portability. zsh-specific scripts may not work on other systems.

2.3 Understanding Paths

Absolute vs Relative Paths

Type	Description	Example
Absolute Path	Full path from root `/`	`/home/kali/Documents/report.txt`
Relative Path	Path from current directory	`./Documents/report.txt` or `../Downloads/`

Special Path Symbols

Symbol	Meaning	Example
`/`	Root directory (top of filesystem)	`cd /`
`~`	Home directory	`cd ~` → `/home/kali`
`.`	Current directory	`./script.sh`
`..`	Parent directory (one level up)	`cd ..`
`-`	Previous directory	`cd -`

2.4 Navigation Commands

Command: `pwd`

Print Working Directory — shows your current location

Syntax:

```
pwd [OPTIONS]
```

Example:

```
┌──(kali㉿kali)-[~]
└─$ pwd
/home/kali

┌──(kali㉿kali)-[/etc/network]
└─$ pwd
/etc/network
```

Options:

Option	Description
`-L`	Print logical path (default, follows symlinks)
`-P`	Print physical path (resolves symlinks)

Real-Life Use Case:
When writing scripts that need to reference files relative to the script's location, `pwd` helps establish the base path. During pentesting, knowing exactly where you are prevents accidentally modifying wrong files.

Command: `cd`

Change Directory — navigate the filesystem

Syntax:

```
cd [DIRECTORY]
```

Examples:

```
cd /var/log          # Go to absolute path
cd Documents         # Go to relative path (inside current dir)
cd ..                # Go up one level
cd ../..             # Go up two levels
cd ~                 # Go to home directory
cd                   # Also goes to home directory
cd -                 # Go to previous directory
```

Pro Techniques:

```
cd /etc/netw[TAB]    # Tab completion → /etc/network
cd ~/Do[TAB][TAB]    # Double-tab shows:  Documents/ Downloads/
```

Real-Life Use Case:
Navigating to `/var/log` to analyze system logs during incident response, or moving to `/etc` to review configuration files during system enumeration.

Command: `ls`

List directory contents — most used command in Linux

Syntax:

```
ls [OPTIONS] [PATH]
```

Common Options:

Option	Description
`-l`	Long format (permissions, size, date)
`-a`	Show hidden files (starting with `.`)
`-la` or `-al`	Long format + hidden files
`-lh`	Long format + human-readable sizes
`-R`	Recursive (list subdirectories)
`-t`	Sort by modification time (newest first)
`-S`	Sort by size (largest first)
`-r`	Reverse sort order
`-1`	One file per line
`--color`	Colorize output (default on most systems)

Examples:

```
┌──(kali㉿kali)-[~]
└─$ ls
Desktop  Documents  Downloads  Music  Pictures  Public

┌──(kali㉿kali)-[~]
└─$ ls -la
total 120
drwxr-xr-x 15 kali kali 4096 Dec 25 10:30 .
drwxr-xr-x  3 root root 4096 Dec 20 09:15 ..
-rw-------  1 kali kali  220 Dec 20 09:15 .bash_history
-rw-r--r--  1 kali kali  220 Dec 20 09:15 . bash_logout
-rw-r--r--  1 kali kali 3526 Dec 20 09:15 . bashrc
drwxr-xr-x  2 kali kali 4096 Dec 20 09:15 Desktop
drwxr-xr-x  2 kali kali 4096 Dec 25 10:30 Documents

┌──(kali㉿kali)-[~]
└─$ ls -lhS /var/log | head -5
total 15M
-rw-r-----  1 root adm  5.2M Dec 25 10:00 syslog
-rw-r-----  1 root adm  2.1M Dec 25 09:00 auth.log
-rw-r-----  1 root adm  1.8M Dec 24 23:59 kern.log
```

Understanding `ls -l` Output:

```
-rw-r--r--  1  kali  kali  4096  Dec 25 10:30  file.txt
|└─┬─┘└─┬┘  |   |     |     |         |           |
|  |    |   |   |     |     |         |           └── Filename
|  |    |   |   |     |     |         └── Modification date
|  |    |   |   |     |     └── Size in bytes
|  |    |   |   |     └── Group owner
|  |    |   |   └── User owner
|  |    |   └── Hard link count
|  |    └── Permissions (rwx for user, group, others)
|  └── File type (- = file, d = directory, l = symlink)
```

Real-Life Use Case:
During enumeration, `ls -la /home` reveals all users on a system. `ls -la` in any directory exposes hidden config files (like `.bashrc`, `.ssh`) that may contain credentials or useful information.

Command: `tree`

Display directory structure as a tree

Syntax:

```
tree [OPTIONS] [PATH]
```

> ⚠ **Note:** May need installation: `sudo apt install tree`

Common Options:

Option	Description
`-L n`	Limit depth to n levels
`-d`	Directories only
`-a`	Include hidden files
`-f`	Print full path
`-h`	Human-readable sizes
`--dirsfirst`	List directories before files

Examples:

```
┌──(kali㉿kali)-[~]
└─$ tree -L 2
.
├── Desktop
│   └── tools
├── Documents
│   ├── notes
│   └── reports
├── Downloads
│   ├── exploits
│   └── wordlists
└── scripts
    ├── recon. sh
    └── scan.sh

┌──(kali㉿kali)-[~]
└─$ tree -L 1 -d /etc
/etc
├── alternatives
├── apache2
├── apt
├── bash_completion. d
├── network
...
```

Real-Life Use Case:
Quickly visualizing web application structures after gaining access. `tree -L 3 /var/www/html` instantly shows the application layout, helping identify config files, upload directories, and potential targets.

2.5 File Operations

Command: `touch`

Create empty files or update timestamps

Syntax:

```
touch [OPTIONS] FILENAME(S)
```

Examples:

```
touch newfile.txt                    # Create single file
touch file1.txt file2.txt file3.txt  # Create multiple files
touch /tmp/test.log                  # Create file in specific location
touch {a,b,c}.txt                    # Create a. txt, b.txt, c.txt
touch file{1..10}.txt                # Create file1.txt through file10.txt
```

Options:

Option	Description
`-a`	Change access time only
`-m`	Change modification time only
`-t STAMP`	Use specific timestamp (format: [[CC]YY]MMDDhhmm[. ss])

Real-Life Use Case:
Creating log files before running scans, or updating timestamps to cover tracks (note: this is detectable with forensic analysis of inode data).

Command: `mkdir`

Make directories

Syntax:

```
mkdir [OPTIONS] DIRECTORY(S)
```

Examples:

```
mkdir projects                       # Create single directory
mkdir dir1 dir2 dir3                 # Create multiple directories
mkdir -p pentest/target/scans/nmap   # Create nested structure (-p = parents)
mkdir -m 700 private                 # Create with specific permissions
mkdir -p project/{src,bin,docs,logs} # Create project structure
```

Options:

Option	Description
`-p`	Create parent directories as needed (no error if exists)
`-m MODE`	Set permissions (like chmod)
`-v`	Verbose output

Real-Life Use Case:
Setting up organized pentest project structures:

```
mkdir -p ~/pentests/client_name/{recon,scans,exploits,loot,reports}
```

Command: `cp`

Copy files and directories

Syntax:

```
cp [OPTIONS] SOURCE DESTINATION
```

Examples:

```
cp file. txt backup. txt             # Copy file
cp file.txt /tmp/                    # Copy to directory
cp file1.txt file2.txt /backup/      # Copy multiple files to directory
cp -r directory/ backup_dir/         # Copy directory recursively
cp -p important.conf important.bak   # Preserve permissions and timestamps
cp -i file.txt /tmp/                 # Interactive (prompt before overwrite)
cp -v *.txt /backup/                 # Verbose output
```

Options:

Option	Description
`-r, -R`	Recursive (required for directories)
`-p`	Preserve permissions, ownership, timestamps
`-i`	Interactive (prompt before overwrite)
`-v`	Verbose
`-n`	No clobber (don't overwrite existing)

Option	Description
`-u`	Update (copy only if source is newer)

Real-Life Use Case:
Backing up configuration files before modifying them during system hardening:

```
cp -p /etc/ssh/sshd_config /etc/ssh/sshd_config.bak
```

Command: `mv`

Move or rename files and directories

Syntax:

```
mv [OPTIONS] SOURCE DESTINATION
```

Examples:

```
mv oldname.txt newname. txt           # Rename file
mv file. txt /tmp/                    # Move to directory
mv file. txt /tmp/newname.txt         # Move and rename
mv *.log /var/log/archive/           # Move multiple files
mv -i important.txt /backup/         # Interactive mode
mv -n file.txt /existing/            # Don't overwrite
```

Options:

Option	Description
`-i`	Interactive (prompt before overwrite)
`-n`	No clobber (don't overwrite)
`-v`	Verbose
`-u`	Update only (move if source is newer)

Real-Life Use Case:
Organizing captured data during a pentest:

```
mv *.pcap ~/pentests/client/captures/
mv scan_results_* ~/pentests/client/scans/
```

Command: `rm`

Remove files and directories

Syntax:

```
rm [OPTIONS] FILE(S)
```

Examples:

```
rm file.txt                          # Remove single file
rm file1.txt file2.txt              # Remove multiple files
rm *.log                            # Remove all . log files
rm -r directory/                    # Remove directory recursively
rm -rf directory/                   # Force remove (no prompts, no errors)
rm -i sensitive.txt                 # Interactive (confirm before delete)
rm -v *.tmp                         # Verbose output
```

Options:

Option	Description
`-r, -R`	Recursive (required for directories)
`-f`	Force (no prompts, ignore nonexistent files)
`-i`	Interactive (prompt before each removal)
`-v`	Verbose

> ⚠ **DANGER:** `rm -rf /` or `rm -rf /*` will destroy your entire system. There is NO undo. Always double-check paths.

> **Safe Practice:** Use `rm -i` or create an alias: `alias rm='rm -i'`

Real-Life Use Case:
Cleaning up temporary files after a scan:

```
rm -rf /tmp/scan_*
rm *.tmp *.bak
```

Command: `rmdir`

Remove empty directories only

Syntax:

```
rmdir [OPTIONS] DIRECTORY(S)
```

Examples:

```
rmdir empty_folder                  # Remove single empty directory
rmdir dir1 dir2 dir3                # Remove multiple empty directories
rmdir -p a/b/c                      # Remove nested empty directories
```

Options:

Option	Description
`-p`	Remove parent directories if they become empty
`-v`	Verbose

> **Note:** `rmdir` only works on EMPTY directories. For non-empty, use `rm -r`.

2.6 Viewing File Contents

Command: `cat`

Concatenate and display file contents

Syntax:

```
cat [OPTIONS] FILE(S)
```

Examples:

```
cat file. txt                        # Display file
cat file1.txt file2.txt              # Display multiple files
cat file1.txt file2.txt > merged.txt # Concatenate into new file
cat -n file.txt                      # Show line numbers
cat -A file.txt                      # Show hidden characters
```

Options:

Option	Description
`-n`	Number all lines
`-b`	Number non-empty lines only
`-A`	Show all (tabs, line endings)
`-s`	Squeeze multiple blank lines

Real-Life Use Case:
Quickly viewing configuration files or combining log files:

```
cat /etc/passwd                     # View user accounts
cat /etc/shadow                     # View password hashes (requires root)
cat access.log error.log > combined.log
```

Command: `less`

View file contents with pagination (scrollable)

Syntax:

```
less [OPTIONS] FILE
```

Navigation Keys:

Key	Action
`Space` / `f`	Forward one page
`b`	Back one page
`g`	Go to beginning
`G`	Go to end
`/pattern`	Search forward
`?pattern`	Search backward
`n`	Next search result
`N`	Previous search result
`q`	Quit

Examples:

```
less /var/log/syslog                # View log file
less +F /var/log/syslog             # Follow mode (like tail -f)
less -N /etc/passwd                 # Show line numbers
```

Real-Life Use Case:
Analyzing large log files during incident response:

```
less /var/log/auth.log              # Review authentication attempts
# Then /Failed to search for failed logins
```

Command: `more`

View file contents with basic pagination (legacy)

Syntax:

```
more [OPTIONS] FILE
```

> Tip: `less` is superior to `more` — it supports backward scrolling. Saying: "less is more"

Command: `head`

Display first lines of a file

Syntax:

```
head [OPTIONS] FILE
```

Examples:

```
head file.txt                       # First 10 lines (default)
head -n 20 file.txt                 # First 20 lines
head -n 5 /etc/passwd               # First 5 lines
head -c 100 file.txt                # First 100 bytes
head -n -5 file.txt                 # All except last 5 lines
```

Options:

Option	Description
`-n NUM`	Print first NUM lines
`-c NUM`	Print first NUM bytes

Real-Life Use Case:
Quickly previewing files without loading entire content:

```
head -n 1 /etc/passwd               # Get root user info
head /var/log/apache2/access.log    # Recent requests
```

Command: `tail`

Display last lines of a file

Syntax:

```
tail [OPTIONS] FILE
```

Examples:

```
tail file.txt                       # Last 10 lines (default)
tail -n 20 file.txt                 # Last 20 lines
tail -f /var/log/syslog             # Follow (real-time updates)
tail -F /var/log/auth.log           # Follow + retry if file rotates
tail -n +5 file.txt                 # Everything from line 5 onwards
```

Options:

Option	Description
`-n NUM`	Print last NUM lines
`-f`	Follow file (real-time monitoring)
	Follow + retry (handles log

Option	Description
`-F`	rotation)
`-c NUM`	Print last NUM bytes

Real-Life Use Case:
Real-time log monitoring during attacks or troubleshooting:

```
tail -f /var/log/auth.log            # Watch login attempts
tail -f /var/log/apache2/access.log  # Watch web traffic
tail -n 50 /var/log/syslog | grep -i error  # Recent errors
```

Key Terms

Term	Definition
Shell	Command interpreter that processes user input and executes commands (bash, zsh)
Terminal Emulator	GUI application that provides access to the shell (GNOME Terminal, Konsole)
Prompt	Text string indicating shell is ready for input. Shows user, host, and current directory
Absolute Path	Full path from root directory (`/home/kali/file.txt`)
Relative Path	Path relative to current directory (`./file.txt`, `../folder/`)
Hidden File	File starting with `.` — hidden from normal `ls` but visible with `ls -a`
Recursive	Operation that applies to directory and all its subdirectories (`-r` or `-R` flag)
Glob/Wildcard	Pattern matching characters (`*` = any chars, `?` = single char)
Symlink	Symbolic link — shortcut pointing to another file or directory
Inode	Data structure storing file metadata (permissions, timestamps, location on disk)

FAQs

Q: What's the difference between `rm` and `rmdir`?

> `rmdir` removes **empty directories only** — a safety feature. `rm -r` removes directories and all contents recursively. Use `rmdir` when you want to ensure you're not accidentally deleting data.

Q: How do I view hidden files?

> Use `ls -a` or `ls -la`. Hidden files start with a dot (`.bashrc`, `.ssh`, `.config`). These are often configuration files or sensitive directories.

Q: What does `cd -` do?

> It returns you to your **previous directory**. Useful for toggling between two locations:
>
> ```
> cd /var/log
> cd /etc/network
> cd -          # Back to /var/log
> cd -          # Back to /etc/network
> ```

Q: How can I create a complex directory structure quickly?

> Use brace expansion:
>
> ```
> mkdir -p project/{src/{main,test},docs,config,logs}
> ```
>
> This creates `project/src/main`, `project/src/test`, `project/docs`, `project/config`, and `project/logs` in one command.

Q: What's safer — `rm` or moving to trash?

> Linux CLI has no trash by default — `rm` is permanent. Options:
>
> - Use `rm -i` for interactive prompts
> - Install `trash-cli`: `sudo apt install trash-cli` then use `trash` instead of `rm`
> - Create alias: `alias rm='rm -i'`

Q: Why does `tail -f` stop updating when a log file rotates?

> Log rotation replaces the file with a new one (different inode). `tail -f` follows the inode, not the filename. Use `tail -F` instead — it follows the filename and retries when rotation occurs.

Practice Tasks

Task 1: Navigate to `/etc` and List Network Configs

Objective: Practice navigation and filtered listing

Steps:

```
cd /etc
ls -la | grep -i network
# OR
ls -la *network* 2>/dev/null
find . -name "*network*" -type f 2>/dev/null
```

Deliverable: List of all files/directories containing "network" in their name under `/etc`

Task 2: Create a Penetration Testing Project Structure

Objective: Practice `mkdir` with brace expansion

Create this structure in one command:

```
~/pentests/
└── project_alpha/
    ├── recon/
    │   ├── passive/
    │   └── active/
    ├── scanning/
    │   ├── nmap/
    │   └── vuln/
    ├── exploitation/
    ├── post-exploitation/
    ├── loot/
    │   ├── credentials/
    │   ├── hashes/
    │   └── files/
    └── reports/
```

Solution:

```
mkdir -p ~/pentests/project_alpha/{recon/{passive,active},scanning/{nmap,vuln},exploitation,post-exploitation,loot/{credentials,hash
```

Verify:

```
tree ~/pentests/project_alpha
```

Deliverable: Screenshot of `tree` output showing the structure

Task 3: Monitor System Logs in Real-Time

Objective: Practice `tail -f` and log analysis

Steps:

1. Open two terminal tabs/windows
2. In Terminal 1:

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

3. In Terminal 2, generate some activity:

```
sudo apt update
logger "Test message from Loki"
```

4. Watch Terminal 1 for the new entries

Bonus Challenge:

```
tail -f /var/log/auth.log | grep -i "failed\|invalid"
```

This monitors for failed login attempts in real-time.

Deliverable: Screenshot showing real-time log entries appearing

---

✅ Module 2 Checkpoint

Before moving to Module 3, confirm you can:

- ☐ Navigate filesystem using `cd` with absolute and relative paths
- ☐ List files with various `ls` options (`-la` , `-lh` , `-R`)
- ☐ Create files with `touch` and directories with `mkdir -p`
- ☐ Copy, move, and delete files/directories safely
- ☐ View files with `cat` , `less` , `head` , `tail`
- ☐ Monitor logs in real-time with `tail -f`
- ☐ Explain the difference between terminal, shell, and console
- ☐ Create complex directory structures with brace expansion

Module 3: File System Mastery

🎯 Introduction

The Linux filesystem is a **hierarchical tree structure** starting from the root `/` . Unlike Windows with its drive letters (C:, D:), Linux mounts everything under a single unified tree. Understanding this structure is essential for navigation, troubleshooting, and **privilege escalation** during penetration testing.

This module covers the directory hierarchy, file types, permissions, ownership, and special permission bits that are critical attack vectors in cybersecurity.

3.1 Linux File System Hierarchy

Everything in Linux starts from `/` (root). Here's the complete breakdown:

```
/
├── bin/       → Essential user binaries (ls, cp, cat)
├── boot/      → Boot loader files (kernel, grub)
├── dev/       → Device files (disks, terminals)
├── etc/       → System configuration files
├── home/      → User home directories
├── lib/       → Essential shared libraries
├── lib64/     → 64-bit libraries
├── media/     → Mount point for removable media
├── mnt/       → Temporary mount points
├── opt/       → Optional/third-party software
├── proc/      → Virtual filesystem (process info)
├── root/      → Root user's home directory
├── run/       → Runtime data (PID files, sockets)
├── sbin/      → System binaries (admin commands)
├── srv/       → Service data (web, ftp)
├── sys/       → Virtual filesystem (kernel/hardware info)
├── tmp/       → Temporary files (cleared on reboot)
├── usr/       → User programs and data
│   ├── bin/   → User binaries
│   ├── lib/   → Libraries
│   ├── local/ → Locally installed software
│   └── share/ → Shared data (docs, icons)
└── var/       → Variable data (logs, mail, spool)
    ├── log/   → Log files
    ├── www/   → Web server files
    └── tmp/   → Persistent temp files
```

Critical Directories for CyberSecurity

Directory	Security Relevance
`/etc/passwd`	User account information (readable by all)
`/etc/shadow`	Password hashes (root only — prime target)
`/etc/sudoers`	Sudo privileges configuration
`/var/log/`	System logs — evidence, audit trails
`/tmp/`	World-writable — malware staging area
`/home/*/`	User data, SSH keys, bash history
`/root/`	Root user's home — highest value target
`/proc/`	Live process info, memory dumps
`/dev/`	Device access — raw disk reads possible

Key Commands for Exploration

```
# View filesystem usage
df -h

# View directory sizes
du -sh /var/log/*

# Find large files
find / -type f -size +100M 2>/dev/null

# View mounted filesystems
mount | column -t
cat /etc/fstab
```

3.2 File Types

Linux treats everything as a file — even devices and processes.

Symbol	Type	Description	Example
`-`	Regular File	Normal files (text, binary, scripts)	`/etc/passwd`
`d`	Directory	Folder containing other files	`/home/kali/`
`l`	Symbolic Link	Pointer to another file	`/bin/sh → /bin/dash`
`c`	Character Device	Character-by-character I/O	`/dev/tty` (terminal)
`b`	Block Device	Block-based I/O (storage)	`/dev/sda` (hard disk)
`p`	Named Pipe (FIFO)	Inter-process communication	Created with `mkfifo`
`s`	Socket	Network/local communication	`/var/run/docker.sock`

Identify File Types

```
# Using ls -l (first character)
ls -la /dev/sda
# brw-rw---- 1 root disk 8, 0 Dec 25 10:00 /dev/sda
# ^-- 'b' = block device

# Using file command
file /etc/passwd
# /etc/passwd: ASCII text

file /bin/ls
# /bin/ls: ELF 64-bit LSB pie executable...

file /dev/null
# /dev/null: character special (1/3)
```

Special Files for Pentesting

File	Purpose	Security Use
`/dev/null`	Discards all input	Suppress output: `cmd 2>/dev/null`
`/dev/zero`	Produces null bytes	Wipe data: `dd if=/dev/zero of=file`

File	Purpose	Security Use
`/dev/urandom`	Random data generator	Generate keys, tokens
`/dev/tcp/host/port`	Bash TCP connection	Reverse shells without netcat

3.3 Hidden Files & Dotfiles

Files starting with `.` are hidden from normal `ls` output.

Common Dotfiles

File	Purpose	Security Value
`~/.bashrc`	Bash configuration	Persistence, aliases, backdoors
`~/.bash_history`	Command history	Leaked credentials, commands
`~/.ssh/`	SSH keys directory	Private keys = full access
`~/.ssh/authorized_keys`	Allowed public keys	Add your key for persistence
`~/.ssh/id_rsa`	Private SSH key	Steal for lateral movement
`~/.profile`	Login shell config	Persistence mechanism
`~/.gitconfig`	Git configuration	May contain credentials
`~/.netrc`	FTP/network credentials	Plaintext passwords
`~/.gnupg/`	GPG keys	Encryption keys

Finding Sensitive Dotfiles

```
# List all hidden files in home
ls -la ~

# Find all .bashrc files on system
find /home -name ". bashrc" 2>/dev/null

# Search for SSH keys
find / -name "id_rsa" 2>/dev/null
find / -name "*. pem" 2>/dev/null

# Check bash history for credentials
cat ~/.bash_history | grep -i "pass\|key\|secret\|token"
```

3.4 File Permissions

Linux uses a **permission matrix** for access control.

Understanding Permission String

```
-rwxr-xr--  1  kali  kali  4096  Dec 25 10:30  script.sh
|├┼┼┼┤
| |  |  |
| |  |  └── Others (everyone else): r-- (read only)
| |  └─────── Group (kali group): r-x (read + execute)
| └────────── User/Owner (kali): rwx (read + write + execute)
└──────────── File type (- = regular file)
```

Permission Values

Symbol	Meaning	Numeric Value
`r`	Read	4
`w`	Write	2
`x`	Execute	1
`-`	No permission	0

Common Permission Sets

Numeric	Symbolic	Meaning
`777`	`rwxrwxrwx`	Full access to everyone (dangerous!)
`755`	`rwxr-xr-x`	Owner full, others read+execute
`700`	`rwx------`	Owner only
`644`	`rw-r--r--`	Owner write, others read only
`600`	`rw-------`	Owner read+write only (SSH keys)
`400`	`r--------`	Owner read only

Command: `chmod`

Change file permissions

Syntax:

```
chmod [OPTIONS] MODE FILE(S)
```

Numeric Method:

```
chmod 755 script. sh      # rwxr-xr-x
chmod 600 id_rsa         # rw------- (SSH private key)
chmod 644 config.txt     # rw-r--r--
chmod 777 /tmp/share     # Full access (dangerous)
```

Symbolic Method:

```
chmod u+x script.sh      # Add execute for user
chmod g-w file.txt       # Remove write from group
chmod o-rwx secret.txt   # Remove all from others
chmod a+r public. txt     # Add read for all
chmod u=rwx,g=rx,o= file # Set exact permissions
```

Symbolic Reference:

Symbol	Meaning
`u`	User (owner)
`g`	Group
`o`	Others

Symbol	Meaning
a	All (u+g+o)
`+`	Add permission
`-`	Remove permission
`=`	Set exact permission

Recursive Permissions:

```
chmod -R 755 /var/www/html/   # Apply to all files/subdirs
```

Real-Life Use Case: Making a script executable before running:

```
chmod +x exploit.sh
./exploit.sh
```

Command: `chown`

Change file owner and group

Syntax:

```
chown [OPTIONS] USER[:GROUP] FILE(S)
```

Examples:

```
chown kali file.txt              # Change owner to kali
chown kali:kali file.txt         # Change owner and group
chown : developers file.txt       # Change group only
chown -R www-data:www-data /var/www/   # Recursive
```

> ⚠ **Note:** Only root can change file ownership.

Real-Life Use Case: Web server files need correct ownership:

```
sudo chown -R www-data:www-data /var/www/html/
```

Command: `chgrp`

Change group ownership

Syntax:

```
chgrp [OPTIONS] GROUP FILE(S)
```

Examples:

```
chgrp developers project/        # Change group
chgrp -R team shared_folder/     # Recursive
```

3.5 Ownership & Groups

Every file has an **owner** and a **group**.

View Ownership

```
ls -l file.txt
# -rw-r--r-- 1 kali kali 4096 Dec 25 10:30 file.txt
#              |    |
#              |    └── Group
#              └── Owner

stat file.txt   # Detailed information
```

User & Group Commands

```
 # View current user
whoami

# View user ID and groups
id
# uid=1000(kali) gid=1000(kali) groups=1000(kali),27(sudo),100(users)

# View all groups
cat /etc/group

# View user info
cat /etc/passwd | grep kali

# View groups user belongs to
groups kali
```

Security Implications

Ownership	Risk
Root-owned files writable by others	Privilege escalation
SUID binaries owned by root	Potential root shell
Misconfigured web directories	Unauthorized file upload
SSH keys with wrong permissions	Key rejected or exposed

3.6 Special Permissions (SUID, SGID, Sticky Bit)

Beyond rwx, Linux has **special permission bits** — major privilege escalation vectors.

SUID (Set User ID)

When executed, runs with **owner's privileges** instead of executor's.

Octal	Symbol	Position
`4000`	`s` or `S`	User execute bit

```
 # Example: passwd command
ls -la /usr/bin/passwd
# -rwsr-xr-x 1 root root 68208 Dec 25 10:30 /usr/bin/passwd
#    ^-- 's' = SUID bit set
```

Why it matters: Regular users can change their password, but `/etc/shadow` is root-only. SUID makes `passwd` run as root temporarily.

Security Risk: If a SUID binary has a vulnerability, attackers can get root shell.

Set SUID:

```
chmod u+s binary         # Symbolic
chmod 4755 binary        # Numeric
```

SGID (Set Group ID)

- On **files:** Executes with group's privileges
- On **directories:** New files inherit directory's group

Octal	Symbol	Position
`2000`	`s` or `S`	Group execute bit

```
ls -la /usr/bin/wall
# -rwxr-sr-x 1 root tty 14488 Dec 25 10:30 /usr/bin/wall
#        ^-- 's' = SGID bit set
```

Set SGID:

```
chmod g+s directory/     # Symbolic
chmod 2755 directory/    # Numeric
```

Sticky Bit

On directories: Only file **owner** can delete their files (even if others have write access).

Octal	Symbol	Position
`1000`	`t` or `T`	Others execute bit

```
ls -ld /tmp
# drwxrwxrwt 15 root root 4096 Dec 25 10:30 /tmp
#          ^-- 't' = sticky bit
```

Why `/tmp` has sticky bit: Everyone can write, but users can't delete each other's files.

Set Sticky Bit:

```
chmod +t directory/      # Symbolic
chmod 1777 directory/    # Numeric
```

Finding SUID/SGID Binaries (Privilege Escalation)

```
# Find all SUID binaries (CRITICAL for pentesting)
find / -perm -4000 -type f 2>/dev/null

# Find all SGID binaries
find / -perm -2000 -type f 2>/dev/null

# Find both SUID and SGID
find / -perm /6000 -type f 2>/dev/null

# Find SUID binaries owned by root
find / -user root -perm -4000 -type f 2>/dev/null

# Common exploitable SUID binaries:
# /usr/bin/find, /usr/bin/vim, /usr/bin/awk
# /usr/bin/python, /usr/bin/perl, /usr/bin/less
```

> **GTFOBins** (https://gtfobins.github.io/) — Database of Unix binaries that can be exploited for privilege escalation.

3.7 Links (Hard vs Symbolic)

Links create references to files.

Symbolic (Soft) Links

Pointer to a filename — like a shortcut.

```
ln -s /path/to/original /path/to/link
```

Characteristics:

- Points to filename (path)
- Can cross filesystems
- Can link to directories
- Breaks if original is deleted/moved
- Has its own inode
- Shows `l` in `ls -l`

Example:

```
ln -s /usr/share/wordlists/rockyou.txt ~/rockyou.txt
ls -la ~/rockyou. txt
# lrwxrwxrwx 1 kali kali 35 Dec 25 10:30 rockyou.txt -> /usr/share/wordlists/rockyou.txt
```

Hard Links

Direct reference to file data (inode).

```
ln /path/to/original /path/to/link
```

Characteristics:

- Points to inode (actual data)
- Cannot cross filesystems
- Cannot link to directories
- Survives if original is deleted
- Shares inode with original
- Shows as regular file in `ls -l`

Example:

```
ln important.txt backup_important.txt
ls -li important.txt backup_important.txt
# 12345 -rw-r--r-- 2 kali kali 1000 Dec 25 important.txt
# 12345 -rw-r--r-- 2 kali kali 1000 Dec 25 backup_important.txt
#  ^-- Same inode         ^-- Link count = 2
```

Comparison Table

Feature	Symbolic Link	Hard Link
Points to	Filename (path)	Inode (data)
Cross filesystem	✅ Yes	❌ No
Link to directories	✅ Yes	❌ No
Original deleted	❌ Breaks (dangling)	✅ Data preserved
Inode	Different	Same

Feature	Symbolic Link	Hard Link
File type	`l` (link)	`-` (regular)
Command	`ln -s target link`	`ln target link`

Key Terms

Term	Definition
Inode	Data structure storing file metadata (permissions, timestamps, disk location). Each file has unique inode number.
SUID	Set User ID — executable runs with owner's privileges. Numeric: 4000.
SGID	Set Group ID — executable runs with group's privileges, or directory inherits group. Numeric: 2000.
Sticky Bit	On directories, only file owner can delete their files. Numeric: 1000.
Symbolic Link	Soft link pointing to a filename. Breaks if target deleted.
Hard Link	Direct reference to inode. Survives original deletion.
Octal Notation	Numeric permission representation (755 = rwxr-xr-x).
umask	Default permission mask for new files (usually 022 → files get 644, dirs get 755).
ACL	Access Control Lists — extended permissions beyond standard user/group/others.

FAQs

Q: Why can't I delete a file even though I have write permission on it?

> You need **write permission on the directory**, not the file. Directory write permission controls adding/removing files. File write permission controls modifying content.

Q: What's the difference between `s` and `S` in permissions?

> Lowercase `s` means SUID/SGID **and** execute permission are set.
> Uppercase `S` means SUID/SGID is set but execute is **not** — usually a misconfiguration.
> `rwSr--r--` = SUID set but not executable (useless).

Q: Why should SSH keys be chmod 600?

> SSH refuses to use keys with loose permissions. `600` (owner read+write only) prevents other users from reading your private key. SSH will error: "Permissions are too open" if not set correctly.

**Q: How do SUID binaries lead to privilege escalation? **

> If a SUID binary owned by root has a vulnerability (command injection, buffer overflow) or allows shell escape (vim, less, find), attackers can spawn a root shell. Check GTFOBins for exploitable binaries.

Q: What happens when I delete the original file of a hard link?

> Nothing — the data remains accessible through the hard link. Data is only deleted when **all** hard links (link count = 0) are removed. That's because hard links point to the inode, not the filename.

Practice Tasks

Task 1: Find All SUID Binaries on Your System

Objective: Identify potential privilege escalation vectors

Commands:

```
# Find all SUID binaries
find / -perm -4000 -type f 2>/dev/null

# Find SUID binaries owned by root (most dangerous)
find / -user root -perm -4000 -type f 2>/dev/null

# Check one against GTFOBins
# Example: if /usr/bin/find has SUID
/usr/bin/find .  -exec /bin/sh -p \; -quit
```

Deliverable:

- List of SUID binaries on your Kali system
- Identify at least 3 binaries listed on GTFOBins
- Document their exploitation method

Task 2: Create a Symbolic Link to a Frequently Used Tool

Objective: Practice symlinks for workflow optimization

Steps:

```
# Create symlink to rockyou wordlist in home
ln -s /usr/share/wordlists/rockyou.txt ~/rockyou

# Create symlink to nmap scripts
ln -s /usr/share/nmap/scripts ~/nmap-scripts

# Verify links
ls -la ~/rockyou ~/nmap-scripts

# Test the link works
head ~/rockyou
ls ~/nmap-scripts | head
```

Deliverable: Screenshot showing symlinks with `ls -la`

Task 3: Set Permissions — Executable Only by You

Objective: Practice chmod for security

Steps:

```
# Create a test script
echo '#!/bin/bash' > ~/myscript.sh
echo 'echo "This is my private script"' >> ~/myscript.sh

# View current permissions
ls -la ~/myscript. sh

# Set:  owner=rwx, group=none, others=none
chmod 700 ~/myscript.sh
# OR
chmod u=rwx,go= ~/myscript. sh

# Verify
ls -la ~/myscript.sh
# -rwx------ 1 kali kali 52 Dec 25 myscript.sh

# Execute
./myscript. sh
```

Bonus Challenge:

```
# Set SUID on the script (educational only)
sudo chmod u+s ~/myscript. sh
ls -la ~/myscript.sh
# What changed? Why is SUID on scripts ignored by Linux?
```

Deliverable:

- Screenshot of script with `700` permissions
- Answer: Why doesn't SUID work on bash scripts?

✅ Module 3 Checkpoint

Before moving to Module 4, confirm:

- ☐ Can explain the purpose of key directories (`/etc`, `/var`, `/tmp`, `/home`)
- ☐ Understand all file types and can identify them with `ls -l`
- ☐ Can set permissions using both numeric and symbolic methods
- ☐ Understand SUID, SGID, and Sticky Bit
- ☐ Can find SUID binaries for privilege escalation
- ☐ Know the difference between hard and symbolic links
- ☐ Understand why file permissions matter for security

Module 4: Text Processing & Manipulation

🎯 Introduction

Text processing is the **backbone of Linux administration and hacking**. Log analysis, data extraction, credential parsing, configuration modification — all rely on text manipulation tools. Mastering `grep`, `sed`, `awk`, and friends transforms you from a Linux user into a Linux power user.

This module covers essential text processing commands that every security professional uses daily for parsing scan results, extracting sensitive data, and automating reconnaissance.

4.1 Basic Text Output

Command: `cat`

Concatenate and display file contents

Covered in Module 2. Additional advanced uses:

```
# Create file with cat (heredoc)
cat << 'EOF' > script.sh
#!/bin/bash
echo "Hello from script"
EOF

# Number lines
cat -n file.txt

# Show hidden characters (tabs, line endings)
cat -A file.txt
# Tabs shown as ^I, line endings as $
```

Command: `echo`

Display text or variables

Syntax:

```
echo [OPTIONS] [STRING]
```

Examples:

```
echo "Hello, World"              # Simple output
echo -n "No newline"             # No trailing newline
echo -e "Line1\nLine2"           # Enable escape sequences
echo -e "Tab:\tHere"             # Tab character
echo $PATH                       # Print variable
echo "User: $(whoami)"           # Command substitution
echo {1..5}                      # Brace expansion:  1 2 3 4 5
```

Escape Sequences (with -e):

Sequence	Meaning
`\n`	Newline
`\t`	Tab
`\r`	Carriage return
`\\`	Backslash
`\033[1;31m`	ANSI color (red)

Real-Life Use Case: Creating quick test files or debugging scripts:

```
echo "admin:password123" >> creds.txt
echo -e "GET / HTTP/1.1\nHost: target.com\n" | nc target.com 80
```

Command: `tee`

Read from stdin, write to stdout AND files

Syntax:

```
command | tee [OPTIONS] FILE(S)
```

Examples:

```
# Save output while still displaying it
ls -la | tee directory_listing.txt

# Append instead of overwrite
echo "new line" | tee -a log.txt

# Write to multiple files
cat data.txt | tee file1.txt file2.txt file3.txt

# Use with sudo (common trick)
echo "new config" | sudo tee /etc/somefile
```

Options:

Option	Description
`-a`	Append to file instead of overwrite

Real-Life Use Case: Save scan results while watching progress:

```
nmap -sV target. com | tee nmap_scan.txt
```

4.2 Searching with grep

Command: `grep`

Search for patterns in text — one of the most important Linux commands.

Syntax:

```
grep [OPTIONS] PATTERN [FILE(S)]
```

Basic Examples:

```
grep "error" /var/log/syslog         # Find lines with "error"
grep -i "error" logfile.txt          # Case-insensitive
grep -v "debug" logfile.txt          # Invert (lines WITHOUT "debug")
grep -n "failed" auth.log            # Show line numbers
grep -c "404" access.log             # Count matches
grep -l "password" *.txt             # List files with matches
grep -r "TODO" /project/             # Recursive search
```

Common Options:

Option	Description
`-i`	Case-insensitive
`-v`	Invert match (exclude pattern)
`-n`	Show line numbers
`-c`	Count matching lines
`-l`	List only filenames
`-L`	List files WITHOUT matches
`-r` / `-R`	Recursive search
`-w`	Match whole words only
`-x`	Match entire line
`-A n`	Show n lines After match
`-B n`	Show n lines Before match
`-C n`	Show n lines Context (before and after)
`-E`	Extended regex (same as `egrep`)
`-o`	Only print matching part
`-P`	Perl-compatible regex (PCRE)

Advanced Examples:

```
# Show context around matches
grep -B 2 -A 2 "error" log.txt       # 2 lines before and after

# Multiple patterns
grep -E "error|warning|critical" log.txt
grep -e "error" -e "warning" log.txt

# Only output the match itself
grep -o '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' log.txt
# Extracts IP addresses
```

Regular Expressions with grep

Basic Regex:

Pattern	Meaning	Example
`.`	Any single character	`a.c` → abc, aXc
`*`	Zero or more of previous	`ab*c` → ac, abc, abbc
`^`	Start of line	`^Error`
`$`	End of line	`failed$`
`[ ]`	Character class	`[aeiou]`
`[^ ]`	Negated class	`[^0-9]`
`\`	Escape special char	`\.txt`

Extended Regex (grep -E or egrep):

Pattern	Meaning	Example
`+`	One or more of previous	`ab+c` → abc, abbc
`?`	Zero or one of previous	`ab?c` → ac, abc
`{n}`	Exactly n times	`a{3}` → aaa
`{n,m}`	Between n and m times	`a{2,4}` → aa, aaa, aaaa
`|`	OR	`cat|dog`
`( )`	Grouping	`(ab)+` → ab, abab

Security-Focused Examples:

```
# Find IP addresses
grep -E -o '([0-9]{1,3}\.){3}[0-9]{1,3}' access.log

# Find email addresses
grep -E -o '[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}' file.txt

# Find URLs
grep -E -o 'https?://[^ ]+' file. txt

# Find failed SSH logins
grep "Failed password" /var/log/auth.log

# Find sudo commands
grep "COMMAND=" /var/log/auth.log
```

Variants: `egrep` and `fgrep`

Command	Equivalent	Description
`egrep`	`grep -E`	Extended regex (supports `+`, `?`, `|`)
`fgrep`	`grep -F`	Fixed strings (no regex, faster for literal search)

```
# Search literal string with special chars
fgrep "price=$100" file.txt    # No need to escape $
```

4.3 Sorting & Filtering

Command: `sort`

Sort lines of text

Syntax:

```
sort [OPTIONS] [FILE]
```

Examples:

```
sort file.txt                  # Alphabetical sort
sort -n numbers.txt            # Numeric sort
sort -r file.txt               # Reverse order
sort -u file.txt               # Remove duplicates while sorting
sort -k 2 file.txt             # Sort by 2nd field
sort -t ":" -k 3 -n /etc/passwd # Sort by UID (3rd field, colon delimiter)
```

Options:

Option	Description
`-n`	Numeric sort
`-r`	Reverse order
`-u`	Unique (remove duplicates)
`-k N`	Sort by Nth field
`-t CHAR`	Field delimiter

Option	Description
`-h`	Human-readable numbers (1K, 2M)
`-f`	Case-insensitive

Real-Life Use Case: Sort IP addresses by request count:

```
cat access.log | cut -d' ' -f1 | sort | uniq -c | sort -rn | head
# Top 10 IPs by request count
```

Command: `uniq`

Report or filter repeated lines (requires sorted input!)

Syntax:

```
uniq [OPTIONS] [INPUT [OUTPUT]]
```

Examples:

```
sort file.txt | uniq             # Remove duplicate lines
sort file.txt | uniq -c          # Count occurrences
sort file.txt | uniq -d          # Show only duplicates
sort file. txt | uniq -u          # Show only unique lines
```

Options:

Option	Description
`-c`	Prefix lines with count
`-d`	Only print duplicates
`-u`	Only print unique lines
`-i`	Case-insensitive

> ⚠ **Critical:** `uniq` only removes **adjacent** duplicates. Always `sort` first!

```
# WRONG:
cat file.txt | uniq

# CORRECT:
cat file. txt | sort | uniq
```

Command: `wc`

Word, line, character, byte count

Syntax:

```
wc [OPTIONS] [FILE(S)]
```

Examples:

```
wc file.txt                     # lines, words, bytes
wc -l file.txt                  # Line count only
wc -w file.txt                  # Word count only
wc -c file.txt                  # Byte count
wc -m file.txt                  # Character count
wc -l *.txt                     # Count lines in multiple files
```

Real-Life Use Case: Count entries in a wordlist:

```
wc -l /usr/share/wordlists/rockyou.txt
# 14344392 /usr/share/wordlists/rockyou.txt
```

Command: `cut`

Extract sections from lines

Syntax:

```
cut [OPTIONS] [FILE]
```

Examples:

```
# Cut by delimiter
cut -d ":" -f 1 /etc/passwd      # Extract usernames
cut -d ":" -f 1,3 /etc/passwd    # Extract username and UID
cut -d ":" -f 1-3 /etc/passwd    # Extract fields 1 through 3

# Cut by character position
cut -c 1-10 file.txt             # First 10 characters
cut -c 5- file.txt               # From 5th character to end

# Extract from CSV
cut -d "," -f 2,4 data.csv       # 2nd and 4th columns
```

Options:

Option	Description
`-d DELIM`	Field delimiter (default: TAB)
`-f N`	Field number(s) to extract
`-c N`	Character position(s)
`--complement`	Invert selection

Real-Life Use Case: Extract usernames and shells from passwd:

```
cut -d ":" -f 1,7 /etc/passwd
# root:/bin/bash
# kali:/bin/zsh
```

Command: `tr`

Translate or delete characters

Syntax:

```
tr [OPTIONS] SET1 [SET2]
```

Examples:

```
# Replace characters
echo "hello" | tr 'a-z' 'A-Z'           # HELLO (lowercase to uppercase)
echo "HELLO" | tr 'A-Z' 'a-z'           # hello (uppercase to lowercase)

# Delete characters
echo "hello 123 world" | tr -d '0-9'    # hello  world
echo "hello" | tr -d 'aeiou'            # hll

# Squeeze repeated characters
echo "hellooooo" | tr -s 'o'            # hello

# Replace newlines with spaces
cat file.txt | tr '\n' ' '

# Replace spaces with newlines (one word per line)
echo "one two three" | tr ' ' '\n'

# ROT13 encoding
echo "secret" | tr 'a-zA-Z' 'n-za-mN-ZA-M'  # frperg
```

Options:

Option	Description
`-d`	Delete characters in SET1
`-s`	Squeeze repeated characters
`-c`	Complement SET1

Character Classes:

Class	Meaning
`[:alpha:]`	All letters
`[:digit:]`	All digits
`[:alnum:]`	Letters and digits
`[:space:]`	Whitespace
`[:lower:]`	Lowercase letters
`[:upper:]`	Uppercase letters

Real-Life Use Case: Clean up data:

```
# Remove non-printable characters
cat dirty.txt | tr -cd '[:print: ]\n'

# Convert Windows line endings to Unix
cat windows.txt | tr -d '\r' > unix.txt
```

4.4 Stream Editing with sed

Command: `sed`

Stream Editor — powerful text transformation tool

Syntax:

```
sed [OPTIONS] 'COMMAND' [FILE]
```

Basic Substitution

```
# Replace first occurrence per line
sed 's/old/new/' file.txt

# Replace ALL occurrences per line (global)
sed 's/old/new/g' file.txt

# Case-insensitive replacement
sed 's/old/new/gi' file.txt

# Replace and save to file
sed -i 's/old/new/g' file.txt        # Edit in-place (careful!)
sed -i. bak 's/old/new/g' file.txt    # Backup before edit
```

Sed Command Structure

```
sed 's/pattern/replacement/flags'
    │  │       │           │
    │  │       │           └── g=global, i=case-insensitive, p=print
    │  │       └── Replacement string
    │  └── Pattern to match (regex)
    └── Substitute command
```

Common sed Operations

```
# Delete lines
sed '/pattern/d' file.txt            # Delete lines matching pattern
sed '5d' file.txt                    # Delete line 5
sed '1,5d' file.txt                  # Delete lines 1-5
sed '/^$/d' file. txt                 # Delete empty lines
sed '/^#/d' file. txt                 # Delete comment lines

# Print specific lines
sed -n '10p' file.txt                # Print only line 10
sed -n '5,10p' file.txt              # Print lines 5-10
sed -n '/error/p' file. txt           # Print lines with "error"

# Insert/Append
sed '3i\New line above' file.txt     # Insert before line 3
sed '3a\New line below' file. txt     # Append after line 3
sed '1i\Header line' file.txt        # Add header

# Multiple commands
sed -e 's/foo/bar/g' -e 's/baz/qux/g' file.txt
sed 's/foo/bar/g; s/baz/qux/g' file.txt
```

Regex with sed

```
# Use capture groups
sed 's/\(.*\)@\(.*\)/User:  \1, Domain: \2/' emails.txt
# bob@example.com → User: bob, Domain: example.com

# Extended regex (-E)
sed -E 's/([0-9]+)/[\1]/g' file.txt
# Wraps numbers in brackets

# Remove HTML tags
sed 's/<[^>]*>//g' page.html

# Extract between patterns
sed -n '/START/,/END/p' file.txt
```

Special Characters

Character	Meaning
`&`	Matched pattern
`\1, \2`	Capture groups
`^`	Start of line
`$`	End of line

```
# Use & to reference match
sed 's/[0-9]*/[&]/g' file.txt    # Wrap numbers in brackets

# Swap order with capture groups
echo "first,last" | sed 's/\(.*\),\(.*\)/\2,\1/'
# last,first
```

Real-Life Use Cases:

```
# Replace http with https
sed -i 's/http: \/\//https:\/\//g' urls.txt
# OR use different delimiter
sed -i 's|http://|https://|g' urls.txt

# Remove trailing whitespace
sed 's/[[:space:]]*$//' file.txt

# Add prefix to lines
sed 's/^/PREFIX:  /' file.txt

# Comment out lines containing pattern
sed '/pattern/s/^/#/' file.txt
```

4.5 Pattern Matching with awk

Command: `awk`

Pattern scanning and processing language — extremely powerful for data extraction.

Syntax:

```
awk [OPTIONS] 'PATTERN { ACTION }' [FILE]
```

awk Basics

Built-in Variables:

Variable	Meaning
`$0`	Entire line
`$1, $2, ...`	Fields (columns)
`NF`	Number of fields
`NR`	Current line number
`FS`	Field separator (default: whitespace)
`OFS`	Output field separator
`RS`	Record separator (default: newline)

Basic Examples

```
# Print specific columns
awk '{print $1}' file.txt            # First column
awk '{print $1, $3}' file.txt        # First and third columns
awk '{print $NF}' file. txt           # Last column

# Custom delimiter
awk -F ":" '{print $1}' /etc/passwd  # Usernames
awk -F "," '{print $2}' data.csv     # Second CSV column

# Print with formatting
awk '{print "User:", $1}' file.txt
awk -F ":" '{print $1 " -> " $7}' /etc/passwd

# Line numbers
awk '{print NR, $0}' file. txt        # Prefix with line number
```

Pattern Matching

```
# Print lines matching pattern
awk '/error/' file.txt               # Lines containing "error"
awk '/^root/' /etc/passwd            # Lines starting with "root"
awk '$3 > 1000' /etc/passwd          # UID > 1000 (using numeric comparison)

# Conditional logic
awk -F ":" '$3 >= 1000 {print $1}' /etc/passwd
# Print usernames with UID >= 1000

# NOT matching
awk '!/comment/' file.txt            # Lines NOT containing "comment"
```

awk Programming

```
# BEGIN and END blocks
awk 'BEGIN {print "Header"} {print $0} END {print "Footer"}' file. txt

# Variables and calculations
awk '{sum += $1} END {print "Total:", sum}' numbers.txt

# Count lines
awk 'END {print NR}' file. txt

# Average
awk '{sum += $1; count++} END {print sum/count}' numbers.txt

# Conditional actions
awk '{if ($3 > 100) print $1, "HIGH"; else print $1, "LOW"}' file.txt
```

Advanced awk Examples

```
# Multiple field separators
awk -F "[,:]" '{print $1, $2}' file.txt

# Print specific line range
awk 'NR>=5 && NR<=10' file.txt

# Remove duplicates (like uniq but doesn't need sorting)
awk '!seen[$0]++' file.txt

# Sum a column
awk -F "," '{sum += $3} END {print sum}' sales.csv

# Print longest line
awk '{if (length > max) {max = length; longest = $0}} END {print longest}' file.txt

# Swap columns
awk '{print $2, $1}' file.txt

# Format output
awk '{printf "%-10s %5d\n", $1, $2}' file.txt
```

Real-Life Use Cases:

```
# Extract usernames and emails from CSV
awk -F "," '{print $1, $3}' users.csv

# Parse Apache logs — count requests per IP
awk '{print $1}' access.log | sort | uniq -c | sort -rn | head

# Find users with bash shell
awk -F ":" '$7 ~ /bash/ {print $1}' /etc/passwd

# Calculate total size of files
ls -l | awk '{sum += $5} END {print "Total:", sum, "bytes"}'

# Extract data between markers
awk '/START/,/END/' file.txt
```

4.6 Comparing Files

Command: `diff`

Compare files line by line

Syntax:

```
diff [OPTIONS] FILE1 FILE2
```

Examples:

```
diff file1.txt file2.txt            # Basic comparison
diff -u file1.txt file2.txt         # Unified format (patch-style)
diff -y file1.txt file2.txt         # Side-by-side
diff -q file1.txt file2.txt         # Just report if different
diff -r dir1/ dir2/                 # Compare directories
```

Output Meaning:

```
< line     # Line only in file1
> line     # Line only in file2
---        # Separator
```

Unified Format (-u):

```
--- file1.txt
+++ file2.txt
@@ -1,3 +1,3 @@
 unchanged line
-removed line
+added line
 unchanged line
```

Options:

Option	Description
`-u`	Unified format (best for patches)
`-y`	Side-by-side
`-q`	Report only whether files differ
`-r`	Recursive (directories)
`-i`	Ignore case
`-w`	Ignore whitespace

Command: `comm`

Compare two sorted files line by line

Syntax:

```
comm [OPTIONS] FILE1 FILE2
```

Output Columns:

- Column 1: Lines only in FILE1
- Column 2: Lines only in FILE2
- Column 3: Lines in both files

Examples:

```
comm file1.txt file2.txt            # All three columns
comm -12 file1.txt file2.txt        # Only common lines (suppress 1,2)
comm -23 file1.txt file2.txt        # Only in file1 (suppress 2,3)
comm -13 file1.txt file2.txt        # Only in file2 (suppress 1,3)
```

> ⚠ **Files must be sorted! ** Use `sort file.txt` first.

Command: `cmp`

Compare two files byte by byte

Syntax:

```
cmp [OPTIONS] FILE1 FILE2
```

Examples:

```
cmp file1.bin file2.bin              # Report first difference
cmp -l file1.bin file2.bin           # List all differences
cmp -s file1.bin file2.bin           # Silent (exit code only)
```

Use Case: Comparing binary files where `diff` isn't useful.

Key Terms

Term	Definition
Regex	Regular Expression — pattern language for text matching. Basic (BRE) vs Extended (ERE).
Pipe	`|` character — connects stdout of one command to stdin of another.
stdin/stdout/stderr	Standard input (0), output (1), and error (2) streams.
Stream	Continuous flow of text data passed between commands.
Field	Column/section of text separated by delimiter (space, comma, colon).
Delimiter	Character that separates fields (`:` in /etc/passwd, `,` in CSV).
In-place Edit	Modifying file directly with `sed -i` (dangerous without backup).
Capture Group	Regex pattern in `\( \)` that can be referenced with `\1, \2`.
Glob	Shell wildcard patterns (`*`, `?`, `[ ]`) — NOT regex.

FAQs

Q: What's the difference between grep regex and sed regex?

> Both use Basic Regular Expressions (BRE) by default. Use `-E` for Extended Regular Expressions (ERE) with both. Key difference: `grep` searches and filters, `sed` searches and transforms.

Q: Why does `uniq` miss some duplicates?

> `uniq` only removes **adjacent** duplicates. You must `sort` first: `sort file.txt | uniq`. This is the #1 mistake with uniq.

Q: When should I use awk vs sed?

> - **sed:** Simple substitutions, delete lines, text transformations
> - **awk:** Column-based data, calculations, complex conditionals, CSV parsing
>
> Rule: If you're working with columns/fields, use awk.

Q: How do I match literal special characters in regex?

> Escape them with backslash: `\.` for period, `\*` for asterisk, `\$` for dollar sign. Or use `fgrep` (grep -F) for literal matching.

Q: What's the difference between `*` in glob vs regex?

> - **Glob (shell):** `*` = any characters (including none) → `*.txt` matches all . txt files
> - **Regex:** `*` = zero or more of *previous* character → `a*` matches "", "a", "aa"

> In regex, "any characters" is `.*` (dot = any char, star = any amount).

Practice Tasks

Task 1: Extract All IP Addresses from a Log File

Objective: Practice grep with regex

Setup:

```
# Create sample log file
cat << 'EOF' > access.log
192.168.1.100 - - [25/Dec/2024:10:30:00] "GET /index.html HTTP/1.1" 200
10.0.0.55 - - [25/Dec/2024:10:31:00] "POST /login HTTP/1.1" 401
172.16.0.1 - - [25/Dec/2024:10:32:00] "GET /admin HTTP/1.1" 403
192.168.1.100 - - [25/Dec/2024:10:33:00] "GET /dashboard HTTP/1.1" 200
8.8.8.8 - - [25/Dec/2024:10:34:00] "GET /api HTTP/1.1" 200
EOF
```

Solution:

```
# Extract IPs
grep -E -o '([0-9]{1,3}\. ){3}[0-9]{1,3}' access. log

# Get unique IPs sorted by frequency
grep -E -o '([0-9]{1,3}\.){3}[0-9]{1,3}' access.log | sort | uniq -c | sort -rn
```

Deliverable: List of unique IPs with request counts

Task 2: Parse CSV — Extract Usernames and Emails

Objective: Practice awk field extraction

Setup:

```
cat << 'EOF' > users.csv
id,username,email,role,created
1,john_doe,john@example. com,admin,2024-01-15
2,jane_smith,jane@example.com,user,2024-02-20
3,bob_wilson,bob@company.org,user,2024-03-10
4,alice_jones,alice@example. com,moderator,2024-04-05
EOF
```

Solution:

```
# Skip header, extract username and email
awk -F "," 'NR > 1 {print $2, $3}' users.csv

# Formatted output
awk -F "," 'NR > 1 {printf "User: %-15s Email: %s\n", $2, $3}' users.csv

# Only admins
awk -F "," '$4 == "admin" {print $2, $3}' users.csv
```

Deliverable: Formatted list of usernames and emails

Task 3: Replace http: // with https:// Using sed

Objective: Practice sed substitution

Setup:

```
cat << 'EOF' > urls.txt
Visit our site at http://example.com
API endpoint: http://api.example.com/v1
Documentation: http://docs.example.com
Secure page: https://secure.example.com
Image: http://cdn.example.com/image.png
EOF
```

Solution:

```
# Preview changes (without -i)
sed 's|http://|https://|g' urls.txt

# Apply changes
sed -i. bak 's|http://|https://|g' urls. txt

# Verify
cat urls.txt
diff urls.txt. bak urls.txt
```

Deliverable:

- Modified urls.txt with all http:// replaced
- Screenshot showing diff output

Module 4 Checkpoint

Before moving to Module 5, confirm:

- ☐ Can use grep with regex for pattern matching
- ☐ Know the difference between grep, egrep, and fgrep
- ☐ Can sort, filter, and count with sort, uniq, wc
- ☐ Can extract fields with cut and awk
- ☐ Can perform substitutions with sed
- ☐ Can parse columnar data with awk
- ☐ Understand when to use sed vs awk
- ☐ Can compare files with diff

Module 5: Redirection & Piping

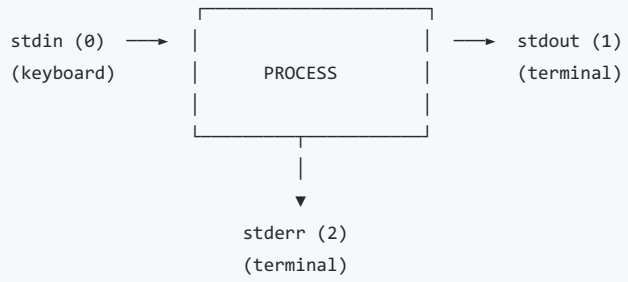
Introduction

Redirection and piping are the **plumbing of Linux** — they control how data flows between commands, files, and the terminal. Mastering these concepts transforms isolated commands into powerful automated workflows.

In cybersecurity, you'll constantly chain tools together: scanning with nmap, filtering with grep, sorting with awk, and saving results — all in one line. This module teaches you to control data flow like a pro.

5.1 Standard Streams

Every Linux process has three default communication channels:



Stream	File Descriptor	Default	Description
stdin	0	Keyboard	Standard input — data fed into command
stdout	1	Terminal	Standard output — normal command output
stderr	2	Terminal	Standard error — error messages

Why Separate Streams?

Separating stdout and stderr allows you to:

- Save output while still seeing errors
- Discard errors while processing output
- Log errors separately for debugging
- Build reliable pipelines

```
 # Both go to terminal by default
ls /home /nonexistent
# Output: lists /home contents
# Error: "ls: cannot access '/nonexistent': No such file or directory"
```

5.2 Output Redirection

Redirect stdout (`>` and `>>`)

Syntax:

```
 command > file      # Overwrite file with stdout
command >> file     # Append stdout to file
```

Examples:

```
 # Overwrite (creates or replaces file)
echo "Hello" > greeting.txt
ls -la > directory_listing.txt
date > timestamp.txt

# Append (adds to end of file)
echo "Line 1" > log.txt
echo "Line 2" >> log.txt
echo "Line 3" >> log.txt
cat log.txt
# Line 1
# Line 2
# Line 3

# Save command output
ps aux > processes.txt
nmap -sV target. com > scan_results.txt
```

> ⚠ **Warning:** `>` overwrites without confirmation! Use `>>` when you want to preserve existing content.

Redirect stderr (`2>` and `2>>`)

Syntax:

```
command 2> file     # Redirect errors to file
command 2>> file    # Append errors to file
```

Examples:

```
# Redirect errors only
ls /nonexistent 2> errors.txt
cat errors.txt
# ls: cannot access '/nonexistent': No such file or directory

# Discard errors (send to /dev/null)
find / -name "*.conf" 2>/dev/null

# Append errors to log
./script.sh 2>> error_log.txt
```

Redirect Both stdout and stderr

Multiple Methods:

```
# Method 1: Separate files
command > output.txt 2> errors.txt

# Method 2: Both to same file (order matters!)
command > all. txt 2>&1

# Method 3: Shorthand (bash 4+)
command &> all.txt
command &>> all.txt    # Append version

# Method 4: stdout to file, stderr to stdout (then to file)
command 2>&1 > file.txt   # WRONG order - stderr goes to terminal
command > file.txt 2>&1   # CORRECT - both to file
```

Understanding `2>&1`:

```
2>&1 means "redirect file descriptor 2 (stderr) to wherever 1 (stdout) is going"
```

Examples:

```
# Save all output including errors
find / -name "*.log" > all_logs.txt 2>&1

# Separate output and errors
./exploit.sh > success.txt 2> failures.txt

# Discard everything
command > /dev/null 2>&1
command &> /dev/null        # Shorthand
```

Redirect stdin (`<`)

Syntax:

```
command < file      # Use file as input instead of keyboard
```

Examples:

```
 # Read input from file
wc -l < /etc/passwd
# Counts lines in passwd file

# Sort file contents
sort < unsorted.txt

# Mail content from file
mail -s "Report" admin@example.com < report. txt

# Use with while loop
while read line; do
    echo "Processing: $line"
done < userlist.txt
```

Here Documents (`<<`)

Feed multi-line input to a command:

```
 # Here document
cat << EOF
This is line 1
This is line 2
Variable expansion: $HOME
EOF

# Here document to file
cat << EOF > config.txt
server=192.168.1.1
port=8080
timeout=30
EOF

# Prevent variable expansion (quote EOF)
cat << 'EOF'
Literal $HOME - no expansion
EOF

# Here string (<<<) - single line
grep "pattern" <<< "search in this string"
wc -w <<< "count these words"
```

Special Destinations

Destination	Description
`/dev/null`	Black hole — discards all input
`/dev/zero`	Produces infinite null bytes
`/dev/urandom`	Produces random data
`/dev/stdin`	Symbolic link to stdin
`/dev/stdout`	Symbolic link to stdout

Destination	Description
`/dev/stderr`	Symbolic link to stderr

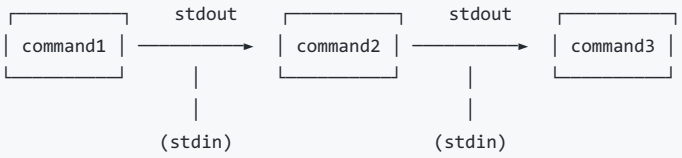
```
# Suppress all output
command > /dev/null 2>&1

# Suppress only errors
command 2>/dev/null

# Generate random password
head -c 16 /dev/urandom | base64 | head -c 16
```

5.3 Piping (|)

The pipe connects stdout of one command to stdin of another.

Syntax:

```
command1 | command2 | command3
```



Basic Piping Examples

```
# Count files
ls -la | wc -l

# Find specific process
ps aux | grep nginx

# Sort and remove duplicates
cat names.txt | sort | uniq

# Show first 10 results
find / -name "*.conf" 2>/dev/null | head -10

# Count unique IPs in log
cat access.log | cut -d' ' -f1 | sort | uniq -c | sort -rn
```

Complex Pipeline Examples

```
# Top 5 largest files in /var
du -ah /var 2>/dev/null | sort -rh | head -5

# Extract emails from file
grep -E -o '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' data.txt | sort -u

# Monitor network connections
netstat -an | grep ESTABLISHED | awk '{print $5}' | cut -d: -f1 | sort | uniq -c | sort -rn

# Parse Apache logs for 404 errors
cat access.log | grep " 404 " | awk '{print $7}' | sort | uniq -c | sort -rn | head -10

# Find SUID binaries and format output
find / -perm -4000 -type f 2>/dev/null | xargs ls -la | awk '{print $1, $9}'
```

Pipe to `xargs`

Convert stdin to command arguments:

```
# Find and delete
find /tmp -name "*.tmp" | xargs rm -f

# Find and grep
find .  -name "*.py" | xargs grep "import os"

# Handle filenames with spaces
find .  -name "*.txt" -print0 | xargs -0 wc -l

# Limit arguments per command
echo "1 2 3 4 5" | xargs -n 2 echo
# 1 2
# 3 4
# 5

# Run with confirmation
find .  -name "*.bak" | xargs -p rm
```

xargs Options:

Option	Description
`-n N`	Use N arguments per command
`-0`	Handle null-delimited input (with find -print0)
`-p`	Prompt before execution
`-I {}`	Replace {} with input

```
# Rename files using -I
find . -name "*.txt" | xargs -I {} mv {} {}. bak

# Download list of URLs
cat urls.txt | xargs -n 1 wget
```

Pipe stderr (`|&`)

Pipe both stdout and stderr:

```
# Bash 4+ shorthand
command |& grep "error"

# Equivalent to:
command 2>&1 | grep "error"
```

5.4 Command Chaining

Sequential Execution (`;`)

Run commands in sequence, regardless of success/failure:

```
command1 ; command2 ; command3
```

Examples:

```
# Always run both
echo "Starting" ; sleep 2 ; echo "Done"

# Run even if first fails
mkdir /invalid 2>/dev/null ; echo "Continuing anyway"

# Multiple operations
cd /var/log ; ls -la ; pwd
```

Conditional AND (`&&`)

Run next command only if previous succeeded (exit code 0):

```
command1 && command2
```

Examples:

```
# Create directory and enter it
mkdir project && cd project

# Update and install
sudo apt update && sudo apt install nmap

# Compile and run
gcc program.c -o program && ./program

# Check file exists, then process
[ -f config.txt ] && source config.txt

# Chain multiple conditions
mkdir dir && cd dir && touch file.txt && echo "All done"
```

Conditional OR (`||`)

Run next command only if previous failed (non-zero exit code):

```
command1 || command2
```

Examples:

```
# Fallback behavior
cat file.txt || echo "File not found"

# Default value
grep "pattern" file. txt || echo "No match found"

# Error handling
ping -c 1 server. com || echo "Server unreachable"

# Create if doesn't exist
[ -d /backup ] || mkdir /backup
```

Combining AND and OR

```
# Try to connect, print status
ping -c 1 server.com && echo "Server UP" || echo "Server DOWN"

# Check service, start if not running
pgrep nginx && echo "Running" || sudo systemctl start nginx

# Conditional execution patterns
[ -f file ] && cat file || echo "No file"
```

Grouping Commands

Use parentheses `()` or braces `{}`:

```
# Subshell (parentheses) — runs in new shell
(cd /tmp && ls)    # Original directory unchanged
pwd                # Still in original directory

# Current shell (braces) — runs in same shell
{ cd /tmp && ls; }  # Directory changes persist
pwd                 # Now in /tmp

# Redirect grouped output
{ echo "Line 1"; echo "Line 2"; } > output.txt

# Conditional groups
[ -f config ] && { source config; echo "Loaded"; } || echo "No config"
```

> **Note:** Braces require space after `{` and semicolon before `}`.

5.5 Subshells & Command Substitution

Command Substitution

Capture command output into variable or use inline:

Syntax:

```
$(command)      # Modern syntax (recommended)
`command`       # Legacy backtick syntax
```

Examples:

```
# Store in variable
current_date=$(date +%Y-%m-%d)
user_count=$(wc -l < /etc/passwd)
my_ip=$(hostname -I | awk '{print $1}')

# Use inline
echo "Today is $(date)"
echo "Current user:  $(whoami)"
echo "Files: $(ls | wc -l)"

# Nested substitution
echo "Kernel:  $(uname -r) on $(uname -m)"

# Create timestamped filename
filename="backup_$(date +%Y%m%d_%H%M%S).tar.gz"

# Dynamic commands
kill $(pgrep nginx)              # Kill all nginx processes
cat $(find .  -name "*.conf")     # Cat all config files
```

Subshells

Run commands in isolated child shell:

```
# Subshell with ()
(cd /tmp && rm -rf tempdir)
# Parent shell's directory unchanged

# Subshell for isolation
(
    export VAR="value"
    echo $VAR
)
echo $VAR    # Empty — variable didn't persist

# Background subshell
(sleep 60 && echo "Done") &
```

Process Substitution

Treat command output as a file:

```
# Compare two command outputs
diff <(ls dir1) <(ls dir2)

# Compare sorted files
diff <(sort file1) <(sort file2)

# Feed multiple inputs
paste <(cut -f1 file1) <(cut -f2 file2)

# Use with commands requiring file argument
./program --config <(echo "setting=value")
```

Key Terms

Term	Definition

Term	Definition
File Descriptor	Integer identifying an I/O stream. 0=stdin, 1=stdout, 2=stderr.
Redirection	Changing where input comes from or output goes to.
Pipe	Connects stdout of one process to stdin of another using `|`.
Exit Code	Number returned by command (0=success, non-zero=error). Check with `$?`.
Subshell	Child shell process created by `()`. Changes don't affect parent.
Command Substitution	Replacing `$(command)` with its output.
Process Substitution	`<(command)` creates pseudo-file from command output.
Here Document	Multi-line input using `<< DELIMITER`.
/dev/null	Special file that discards all data written to it.

❓ FAQs

Q: What's the difference between `>` and `>>`?

> `>` **overwrites** the file (creates new or replaces content).
> `>>` **appends** to the file (adds to end, preserves existing content).
> Common mistake: Using `>` in a loop wipes the file each iteration.

Q: Why doesn't `command > file 2>&1` work when written as `command 2>&1 > file`?

> Order matters! Redirections are processed left to right.
>
> - `> file 2>&1`: stdout goes to file, then stderr goes wherever stdout is (file).
> - `2>&1 > file`: stderr goes wherever stdout is (terminal), then stdout goes to file.
>
> The second form sends stderr to terminal, not file.

Q: How do I check if a command succeeded?

> Check the exit code with `$?`:
>
> ```
> command
> echo $?    # 0 = success, non-zero = failure
> ```
>
> Or use `&&` and `||` for conditional execution.

Q: What's the difference between `;` and `&&`?

> `;` runs commands sequentially **regardless** of success.
> `&&` runs next command **only if** previous succeeded.
> Use `&&` when commands depend on each other, `;` for independent commands.

Q: When should I use `xargs` vs a simple pipe?

> Use `xargs` when the receiving command doesn't read from stdin but expects arguments:
>
> ```
> # rm doesn't read stdin, needs arguments
> find . -name "*. tmp" | xargs rm     # Works
> find . -name "*.tmp" | rm           # Fails
>
> # grep reads stdin
> cat file | grep pattern             # Works
> ```

🛠️ Practice Tasks

Task 1: Redirect stdout and stderr Separately

Objective: Understand stream separation

Setup and Execution:

```
# Create a command that produces both output and errors
ls /home /nonexistent /tmp /invalid 2> errors.txt > success.txt

# Verify
echo "=== Success (stdout) ==="
cat success.txt

echo "=== Errors (stderr) ==="
cat errors. txt
```

Alternative with find:

```
find /etc -name "*.conf" > found_configs.txt 2> permission_errors.txt
wc -l found_configs.txt permission_errors.txt
```

Deliverable:

- Two files: one with successful output, one with errors
- Screenshot showing contents of both files

Task 2: Check Service and Start If Not Running

Objective: Practice conditional chaining

Solution:

```
# Check if Apache is running, start if not
pgrep apache2 && echo "Apache is running" || sudo systemctl start apache2

# More robust version with status check
systemctl is-active --quiet apache2 && echo "Running" || {
    echo "Apache not running, starting..."
    sudo systemctl start apache2
}

# Generic pattern for any service
SERVICE="ssh"
pgrep -x $SERVICE > /dev/null && echo "$SERVICE is running" || {
    echo "$SERVICE not running, attempting start..."
    sudo systemctl start $SERVICE
}
```

Alternative with if-else:

```
if pgrep nginx > /dev/null; then
    echo "nginx is running"
else
    echo "nginx is not running, starting..."
    sudo systemctl start nginx
fi
```

Deliverable: Screenshot of command execution showing conditional behavior

---

Task 3: Find Top 5 Largest Files in /var

Objective: Practice piping for data analysis

Solution:

```
# Method 1: Using du and sort
sudo du -ah /var 2>/dev/null | sort -rh | head -5

# Method 2: Find files only (not directories)
sudo find /var -type f -exec du -h {} + 2>/dev/null | sort -rh | head -5

# Method 3: More detailed with ls format
sudo find /var -type f -printf '%s %p\n' 2>/dev/null | sort -rn | head -5 | awk '{printf "%. 2f MB  %s\n", $1/1024/1024, $2}'

# Save results to file while viewing
sudo du -ah /var 2>/dev/null | sort -rh | head -5 | tee /tmp/large_files.txt
```

Expected Output:

```
 1.2G    /var/log/journal/abc123/system.journal
500M    /var/cache/apt/archives/package. deb
256M    /var/lib/docker/overlay2/...
...
```

Deliverable: List of top 5 largest files with sizes

✅ Module 5 Checkpoint

Before moving to Module 6, confirm:

- ☐ Understand stdin, stdout, stderr and their file descriptors
- ☐ Can redirect output to files (`>`, `>>`)
- ☐ Can redirect errors separately (`2>`, `2>&1`)
- ☐ Can pipe commands together effectively
- ☐ Know the difference between `;`, `&&`, and `||`
- ☐ Can use command substitution `$()`
- ☐ Understand when to use xargs with pipes

Module 6: Process Management

🎯 Introduction

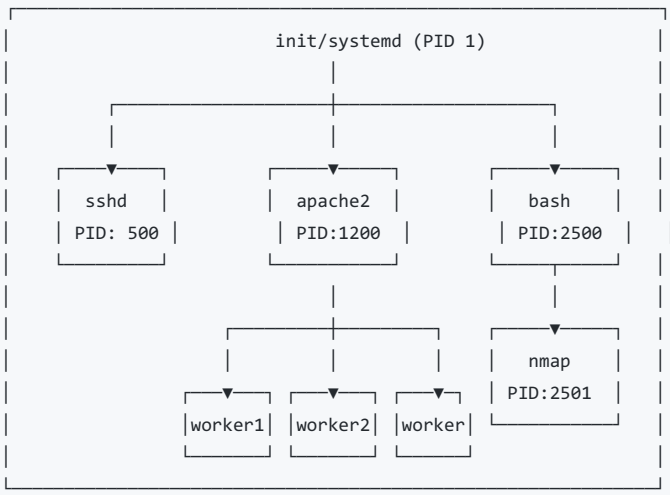
A **process** is any running instance of a program. Understanding processes is crucial for system administration, troubleshooting, and cybersecurity — you need to identify malicious processes, manage resource-hungry applications, and control background tasks.

This module covers viewing, controlling, and terminating processes — essential skills for incident response and system optimization.

6.1 What is a Process?

When you execute a program, the kernel creates a **process** — an instance of that program in memory with its own:

- **PID (Process ID):** Unique identifier
- **PPID (Parent PID):** ID of the process that spawned it
- **UID:** User who owns the process
- **Memory space:** Allocated RAM
- **State:** Running, sleeping, stopped, zombie
- **Priority:** CPU scheduling priority



Process States

State	Symbol	Description
Running	`R`	Currently executing on CPU
Sleeping	`S`	Waiting for event (interruptible)
Disk Sleep	`D`	Waiting for I/O (uninterruptible)
Stopped	`T`	Stopped by signal (Ctrl+Z)
Zombie	`Z`	Finished but parent hasn't collected exit status

6.2 Viewing Processes

Command: `ps`

Process Status — snapshot of current processes

Syntax:

```
ps [OPTIONS]
```

Common ps Usages

```
# Basic — processes in current terminal
ps
#   PID TTY          TIME CMD
#  2500 pts/0    00:00:00 bash
#  2501 pts/0    00:00:00 ps

# All processes for current user
ps -u $(whoami)

# All processes on system (BSD style)
ps aux

# All processes on system (UNIX style)
ps -ef

# Full format with hierarchy
ps -ejH
ps axjf

# Specific columns
ps -eo pid,ppid,user,%cpu,%mem,cmd
```

Understanding `ps aux` Output

```
ps aux
# USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
# root         1  0.0  0.1 169836 11984 ?        Ss   Dec25   0:02 /sbin/init
# kali      2500  0.0  0.0   8024  4532 pts/0    Ss   10:00   0:00 -bash
```

Column	Description
`USER`	Process owner
`PID`	Process ID
`%CPU`	CPU usage percentage
`%MEM`	Memory usage percentage
`VSZ`	Virtual memory size (KB)
`RSS`	Resident Set Size — actual RAM used (KB)
`TTY`	Terminal (? = no terminal)
`STAT`	Process state
`START`	Start time
`TIME`	Total CPU time
`COMMAND`	Command that started process

STAT Column Codes

Code	Meaning
`R`	Running
`S`	Sleeping (interruptible)
`D`	Disk sleep

Code	Meaning
	(uninterruptible)
`T`	Stopped
`Z`	Zombie
`<`	High priority
`N`	Low priority
`s`	Session leader
`+`	Foreground process group
`l`	Multi-threaded

Command: `top`

Real-time process viewer

Syntax:

```
top [OPTIONS]
```

Display:

```
top - 10:30:00 up 5 days,  2:15,  2 users,  load average: 0.15, 0.10, 0.05
Tasks:  215 total,   1 running, 214 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.0 us,  1.0 sy,  0.0 ni, 96.5 id,  0.5 wa,  0.0 hi,  0.0 si
MiB Mem :   15884. 5 total,  10234.2 free,   3256.8 used,   2393.5 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   12123.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1234 root      20   0  256788  45632  12456 S   5.0   0.3   0:15.20 apache2
 5678 kali      20   0   85432  32100   8700 R   2.5   0.2   0:05.10 nmap
```

top Interactive Commands

Key	Action
`h`	Help
`q`	Quit
`k`	Kill process (prompts for PID)
`r`	Renice process (change priority)
`P`	Sort by CPU usage
`M`	Sort by memory usage
`N`	Sort by PID
`T`	Sort by time
`c`	Toggle full command path
`1`	Toggle per-CPU stats
`u`	Filter by user
`Space`	Refresh immediately

Key	Action

Useful Options:

```
top -u kali            # Show only kali's processes
top -p 1234,5678       # Monitor specific PIDs
top -n 1 -b            # Batch mode (single snapshot)
top -d 2               # Update every 2 seconds
```

Command: `htop`

Enhanced interactive process viewer (more user-friendly than top)

```
# Install if needed
sudo apt install htop

htop
```

Features:

- Colorful interface
- Mouse support
- Horizontal and vertical scrolling
- Easy process killing (F9)
- Tree view (F5)
- Search (F3)
- Filter (F4)
- Sort by any column

Command: `pstree`

Display process tree

Syntax:

```
pstree [OPTIONS] [PID|USER]
```

Examples:

```
# Full tree
pstree

# Show PIDs
pstree -p

# Show specific user's processes
pstree kali

# Show ancestry of specific PID
pstree -s -p 1234

# Compact view
pstree -c
```

Sample Output:

```
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─apache2───5*[apache2]
        ├─sshd───sshd───bash───pstree
        └─systemd-journal
```

Other Useful Process Commands

```
# Find process by name
pgrep nginx              # Returns PID(s)
pgrep -l nginx           # With process name
pgrep -u kali            # By user

# Find process using port
lsof -i : 80              # What's using port 80
ss -tulpn | grep :80     # Alternative
netstat -tulpn | grep : 80

# Find process using file
lsof /var/log/syslog

# List open files by process
lsof -p 1234

# Count processes
ps aux | wc -l
pgrep -c bash            # Count bash processes
```

6.3 Background & Foreground Jobs

Running in Background (`&`)

```
# Start in background
nmap -sV target.com &
# [1] 12345       ← Job number [1], PID 12345

# Long-running task in background
./scan_network.sh &

# Redirect output when backgrounding
nmap -sV target.com > scan. txt 2>&1 &
```

Command: `jobs`

List background jobs

```
jobs
# [1]+  Running                 nmap -sV target.com &
# [2]-  Stopped                 vim file.txt

jobs -l      # Include PIDs
jobs -r      # Only running
jobs -s      # Only stopped
```

Job Status:

Status	Meaning
`Running`	Executing in background
`Stopped`	Suspended (Ctrl+Z)
`Done`	Completed
`Terminated`	Killed

Command: `fg`

Bring job to foreground

```
fg           # Bring most recent job to foreground
fg %1        # Bring job [1] to foreground
fg %nmap     # Bring job starting with "nmap"
```

Command: `bg`

Resume stopped job in background

```
# 1. Start a command
vim file.txt

# 2. Suspend with Ctrl+Z
# [1]+  Stopped                 vim file. txt

# 3. Resume in background
bg %1
```

Job Control Shortcuts

Shortcut	Action
`Ctrl+C`	Kill foreground process (SIGINT)
`Ctrl+Z`	Suspend foreground process (SIGTSTP)
`Ctrl+D`	End of input (EOF)

Disown and Nohup

```
# Prevent job from dying when terminal closes
nohup ./long_script.sh &
# Output goes to nohup. out

# Disown a running job (remove from shell's job table)
./script.sh &
disown %1
# Now script survives terminal close

# Disown all background jobs
disown -a
```

6.4 Signals

Signals are **software interrupts** sent to processes.

Common Signals

Signal	Number	Action	Description
`SIGHUP`	1	Terminate	Hangup (terminal closed)

Signal	Number	Action	Description
`SIGINT`	2	Terminate	Interrupt (Ctrl+C)
`SIGQUIT`	3	Core dump	Quit (Ctrl+)
`SIGKILL`	9	Terminate	Kill (cannot be caught/ignored)
`SIGTERM`	15	Terminate	Graceful termination (default)
`SIGSTOP`	19	Stop	Stop process (cannot be caught)
`SIGCONT`	18	Continue	Resume stopped process
`SIGTSTP`	20	Stop	Terminal stop (Ctrl+Z)

Command: `kill`

Send signal to process

Syntax:

```
kill [SIGNAL] PID(S)
```

Examples:

```
# Graceful termination (default:  SIGTERM)
kill 1234

# Force kill (SIGKILL)
kill -9 1234
kill -KILL 1234
kill -SIGKILL 1234

# Multiple processes
kill 1234 5678 9012

# Send specific signal
kill -HUP 1234       # Often used to reload config
kill -STOP 1234      # Pause process
kill -CONT 1234      # Resume process

# List all signals
kill -l
```

> ⚠ **SIGTERM vs SIGKILL:**
>
> - `SIGTERM (15)` : Asks process to terminate gracefully — process can clean up
> - `SIGKILL (9)` : Forces immediate termination — process cannot catch or ignore

---

Command: `killall`

Kill processes by name

Syntax:

```
killall [OPTIONS] PROCESS_NAME
```

Examples:

```
killall firefox            # Kill all firefox processes
killall -9 python          # Force kill all python processes
killall -u kali            # Kill all processes owned by kali
killall -i nginx           # Interactive (confirm each)
killall -w script.sh       # Wait for processes to die
```

Command: `pkill`

Kill processes matching pattern

Syntax:

```
pkill [OPTIONS] PATTERN
```

Examples:

```
pkill nginx               # Kill processes matching "nginx"
pkill -9 -f "python scan"  # Force kill matching full command
pkill -u kali              # Kill all by user
pkill -t pts/0             # Kill processes on terminal pts/0
pkill -P 1234              # Kill child processes of PID 1234
```

Options:

Option	Description
`-f`	Match against full command line
`-u USER`	Match by user
`-t TERM`	Match by terminal
`-P PPID`	Match by parent PID
`-9`	Send SIGKILL

6.5 Process Priority

Linux uses **nice values** to determine CPU scheduling priority.

Nice Value Range

Nice Value	Priority
`-20`	Highest priority
`0`	Default priority
`19`	Lowest priority

> Lower nice value = higher priority (less "nice" to other processes)

Command: `nice`

Start process with modified priority

Syntax:

```
nice -n VALUE COMMAND
```

Examples:

```
# Low priority (be nice to others)
nice -n 10 ./heavy_script.sh

# High priority (requires root)
sudo nice -n -10 ./important_process

# Default nice value (10)
nice ./script.sh
```

Command: `renice`

Change priority of running process

Syntax:

```
renice PRIORITY -p PID(S)
renice PRIORITY -u USER
```

Examples:

```
# Lower priority of running process
renice 15 -p 1234

# Higher priority (root required)
sudo renice -10 -p 1234

# Change all processes for user
sudo renice 5 -u kali
```

View Priority

```
# In ps
ps -eo pid,ni,cmd | head

# In top
top    # NI column shows nice value

# Specific process
ps -o pid,ni,cmd -p 1234
```

6.6 Daemons & Services

Daemons are background processes that run continuously, typically started at boot. They handle system services (web servers, SSH, databases).

Service Management with systemctl

```
# Check service status
systemctl status nginx
systemctl status sshd

# Start/stop/restart
sudo systemctl start nginx
sudo systemctl stop nginx
sudo systemctl restart nginx

# Reload config (without full restart)
sudo systemctl reload nginx

# Enable/disable at boot
sudo systemctl enable nginx
sudo systemctl disable nginx

# Check if enabled
systemctl is-enabled nginx

# Check if active
systemctl is-active nginx

# List all services
systemctl list-units --type=service
systemctl list-units --type=service --state=running

# View service logs
journalctl -u nginx
journalctl -u nginx --since "1 hour ago"
journalctl -u nginx -f    # Follow logs
```

Common Services for CyberSec

Service	Description	Port
`ssh` / `sshd`	SSH server	22
`apache2` / `httpd`	Web server	80, 443
`nginx`	Web server / reverse proxy	80, 443
`mysql` / `mariadb`	Database	3306
`postgresql`	Database	5432
`docker`	Container engine	—
`smbd`	Samba file sharing	445

Key Terms

Term	Definition
Process	Running instance of a program with its own memory space and PID.
PID	Process ID — unique identifier for each process.
PPID	Parent Process ID — PID of the process that spawned this one.
Daemon	Background process running continuously, usually started at boot.

Term	Definition
Signal	Software interrupt sent to process (SIGTERM, SIGKILL, etc.).
Zombie	Process finished but parent hasn't collected exit status.
Nice Value	CPU scheduling priority (-20 to 19). Lower = higher priority.
Foreground	Process attached to terminal, receives keyboard input.
Background	Process running without terminal attachment.
Job	Shell's reference to a process/pipeline it started.

❓ FAQs

Q: What's the difference between kill and kill -9?

> `kill` sends SIGTERM (15) — asks process to terminate gracefully, allowing cleanup.
> `kill -9` sends SIGKILL — forces immediate termination, cannot be caught or ignored.
> Always try SIGTERM first; use SIGKILL only if process won't stop.

Q: How do I find which process is using a specific port?

> Multiple methods:
>
> ```
> lsof -i : 80
> ss -tulpn | grep :80
> netstat -tulpn | grep :80
> fuser 80/tcp
> ```

Q: What's a zombie process and how do I kill it?

> Zombie is a finished process whose parent hasn't collected its exit status. You can't kill zombies directly — they're already dead! Either:
>
> 1. Kill the parent process (zombie will be cleaned up)
> 2. Wait for parent to call `wait()`
> 3. Reboot if zombies are orphaned

Q: Why can't I use nice values below 0?

> Negative nice values (higher priority) require root privileges. This prevents regular users from starving others of CPU time.
>
> ```
> sudo nice -n -10 ./important_task
> ```

Q: How do I keep a process running after I close SSH?

> Several methods:
>
> ```
> nohup ./script. sh &           # Ignores hangup signal
> screen -S session ./script.sh # Screen session
> tmux new -s session           # Tmux session
> disown %1                     # Remove from shell job control
> ```

🛠️ Practice Tasks

Task 1: Identify and Kill a High CPU Process

Objective: Practice process identification and termination

Scenario: Find the process consuming the most CPU and terminate it.

Steps:

```
 # Create a CPU-intensive process (for testing)
yes > /dev/null &
# [1] 12345

# Method 1: Using top
top
# Press 'P' to sort by CPU
# Note the PID of highest CPU process
# Press 'k', enter PID, confirm

# Method 2: Using ps
ps aux --sort=-%cpu | head -5
# Note the PID

# Kill the process
kill 12345
# OR force kill
kill -9 12345

# Verify it's gone
ps aux | grep 12345
```

Alternative — One-liner:

```
 # Find and kill highest CPU process (careful!)
kill $(ps aux --sort=-%cpu | awk 'NR==2 {print $2}')
```

Deliverable: Screenshot of top/ps showing high CPU process, then proof of termination

---

Task 2: Run Scan in Background, Bring to Foreground

Objective: Practice job control

Steps:

```
 # Start nmap scan in background
nmap -sV -p 1-1000 scanme.nmap. org > scan_results.txt 2>&1 &
# [1] 15678

# Check job status
jobs
# [1]+  Running    nmap -sV -p 1-1000 scanme. nmap.org > scan_results.txt 2>&1 &

# Monitor progress
tail -f scan_results.txt &

# Bring nmap to foreground
fg %1

# Suspend it (Ctrl+Z)
# [1]+  Stopped    nmap -sV -p 1-1000 scanme. nmap.org > scan_results.txt

# Resume in background
bg %1

# Check jobs again
jobs -l
```

Deliverable: Screenshot showing job transitions (Running → Stopped → Running)

Task 3: Find All Processes Owned by Specific User

Objective: Practice process filtering (useful for incident response)

Steps:

```
 # Method 1: ps with user filter
ps -u kali
ps aux | grep "^kali"

# Method 2: Using pgrep
pgrep -u kali
pgrep -u kali -l    # With names

# Method 3:  Detailed view
ps -u kali -o pid,ppid,%cpu,%mem,stat,cmd

# Count processes
ps -u kali | wc -l
pgrep -c -u kali

# Method 4: Using top
top -u kali

# Find processes for root (security check)
ps aux | awk '$1=="root"' | wc -l
```

For incident response — find suspicious processes:

```
 # Processes with network connections
lsof -i -u kali

# Processes running from /tmp (often malicious)
ps aux | grep "/tmp/"

# Hidden processes (names starting with .)
ps aux | grep " \./\."
```

Deliverable: List of all processes for a specific user with PID, CPU%, MEM%, and command

---

Module 6 Checkpoint

Before moving to Module 7, confirm:

- ☐ Can view processes with ps, top, htop
- ☐ Understand PID, PPID, and process states
- ☐ Can run processes in background and manage with jobs, fg, bg
- ☐ Know the difference between SIGTERM and SIGKILL
- ☐ Can use kill, killall, and pkill to terminate processes
- ☐ Understand nice values and process priority
- ☐ Can manage services with systemctl
- ☐ Can identify processes by user, port, or resource usage

Module 7: User & Group Management

Introduction

Linux is a **multi-user operating system** — multiple users can work simultaneously with isolated environments and permissions. Understanding user and group management is essential for system administration, access control, and **privilege escalation** during penetration testing.

This module covers creating users, managing groups, configuring sudo access, and understanding the files that control authentication — critical knowledge for both defenders and attackers.

7.1 Users & Root

Types of Users

Type	UID Range	Description
Root	0	Superuser — unrestricted access to everything
System Users	1-999	Service accounts (apache, mysql, nobody)
Regular Users	1000+	Normal human users

Key User Files

File	Purpose	Permission
`/etc/passwd`	User account information	World-readable
`/etc/shadow`	Encrypted passwords	Root only
`/etc/group`	Group information	World-readable
`/etc/gshadow`	Group passwords	Root only
`/etc/sudoers`	Sudo configuration	Root only
`/etc/login.defs`	Default login settings	Root only

`/etc/passwd` Structure

```
cat /etc/passwd
# root:x:0:0:root:/root:/bin/bash
# kali:x:1000:1000:Kali User,,,:/home/kali:/bin/zsh
```

Format:

```
username:password: UID:GID: GECOS:home_directory:shell
   |        |     |   |    |         |           |
   |        |     |   |    |         |           └── Login shell
   |        |     |   |    |         └── Home directory
   |        |     |   |    └── Comment (full name, phone, etc.)
   |        |     |   └── Primary Group ID
   |        |     └── User ID
   |        └── 'x' means password in /etc/shadow
   └── Username
```

Extract Useful Info:

```
# List all usernames
cut -d: -f1 /etc/passwd

# Users with shell access
grep -v "nologin\|false" /etc/passwd | cut -d: -f1

# Users with UID >= 1000 (regular users)
awk -F: '$3 >= 1000 {print $1}' /etc/passwd

# Find user's shell
grep "^kali:" /etc/passwd | cut -d: -f7
```

`/etc/shadow` Structure

```
sudo cat /etc/shadow
# root:$y$j9T$abc123.. .:19500:0:99999:7:::
# kali: $y$j9T$xyz789...:19520:0:99999:7:::
```

Format:

```
username:password_hash:lastchange:min: max:warn:inactive:expire: reserved
   |         |            |       |   |    |      |        |
   |         |            |       |   |    |      |        └── Account expiry
   |         |            |       |   |    |      └── Days after expire until disabled
   |         |            |       |   |    └── Days before expire to warn
   |         |            |       |   └── Max days between changes
   |         |            |       └── Min days between changes
   |         |            └── Days since Jan 1, 1970 of last change
   |         └── Encrypted password (or !, *, empty)
   └── Username
```

Password Field Values:

Value	Meaning
`$y$...` , `$6$...`	Encrypted password (yescrypt, SHA-512)
`!` or `!!`	Account locked
`*`	No password login (system account)
Empty	No password required (dangerous!)

Hash Types:

Prefix	Algorithm
`$1$`	MD5 (weak, deprecated)
`$5$`	SHA-256
`$6$`	SHA-512
`$y$`	yescrypt (modern, Kali default)

7.2 Adding, Modifying, Deleting Users

Command: `useradd`

Create new user account

Syntax:

```
useradd [OPTIONS] USERNAME
```

Examples:

```
# Basic user (no home, no password)
sudo useradd testuser

# Full user setup
sudo useradd -m -s /bin/bash -c "Test User" testuser
#             |    |           |
#             |    |           └── Comment (full name)
#             |    └── Shell
#             └── Create home directory

# Specify UID and GID
sudo useradd -m -u 1500 -g users testuser

# Add to multiple groups
sudo useradd -m -G sudo,docker,www-data newuser

# Set expiry date
sudo useradd -m -e 2025-12-31 tempuser

# Create system user (for services)
sudo useradd -r -s /usr/sbin/nologin serviceuser
```

Common Options:

Option	Description
`-m`	Create home directory
`-M`	Do NOT create home directory
`-s SHELL`	Login shell
`-c COMMENT`	Full name / description
`-d DIR`	Home directory path
`-g GROUP`	Primary group
`-G GROUPS`	Supplementary groups (comma-separated)
`-u UID`	User ID
`-e DATE`	Account expiry (YYYY-MM-DD)
`-r`	Create system account

> ⚠ **Note:** `useradd` does NOT set a password. Use `passwd` afterward.

Command: `adduser` (Debian/Ubuntu/Kali)

Interactive user creation (friendlier wrapper around useradd)

```
sudo adduser newuser
# Prompts for password, full name, etc.
# Automatically creates home directory
```

Command: `usermod`

Modify existing user

Syntax:

```
usermod [OPTIONS] USERNAME
```

Examples:

```
 # Change shell
sudo usermod -s /bin/zsh kali

# Change home directory
sudo usermod -d /home/newhome -m kali
#                             └── Move contents to new home

# Add to group (append!)
sudo usermod -aG sudo kali
sudo usermod -aG docker,wireshark kali
#             └── IMPORTANT: -a = append, without it replaces all groups!

# Change username
sudo usermod -l newname oldname

# Lock account
sudo usermod -L username

# Unlock account
sudo usermod -U username

# Set expiry
sudo usermod -e 2025-06-30 tempuser

# Remove expiry
sudo usermod -e "" username
```

Critical Options:

Option	Description
`-aG GROUP`	**Append** to supplementary group(s) — ALWAYS use `-a` with `-G`!
`-g GROUP`	Change primary group
`-l NAME`	Change login name
`-L`	Lock account
`-U`	Unlock account
`-s SHELL`	Change shell
`-d DIR`	Change home directory
`-m`	Move home contents (use with `-d`)
`-e DATE`	Set expiry date

> ⚠ **Warning:** `usermod -G group user` WITHOUT `-a` **replaces** all supplementary groups!

Command: `userdel`

Delete user account

Syntax:

```
userdel [OPTIONS] USERNAME
```

Examples:

```
# Delete user only (keep home and mail)
sudo userdel testuser

# Delete user AND home directory
sudo userdel -r testuser

# Force delete (even if logged in)
sudo userdel -f testuser
```

Options:

Option	Description
`-r`	Remove home directory and mail spool
`-f`	Force removal even if user is logged in

7.3 Managing Groups

Viewing Groups

```
# List all groups
cat /etc/group

# View user's groups
groups
groups kali
id kali

# View group members
getent group sudo
grep "^sudo:" /etc/group
```

`/etc/group` Structure

```
groupname:password:GID:members
sudo:x:27:kali,admin
```

Command: `groupadd`

Create new group

Syntax:

```
groupadd [OPTIONS] GROUPNAME
```

Examples:

```
# Create group
sudo groupadd developers

# Specify GID
sudo groupadd -g 2000 developers

# Create system group
sudo groupadd -r appgroup
```

Command: `groupdel`

Delete group

```
sudo groupdel developers
```

> ⚠ Cannot delete a group that is a user's primary group.

Command: `groupmod`

Modify group

```
# Rename group
sudo groupmod -n newname oldname

# Change GID
sudo groupmod -g 3000 groupname
```

Command: `gpasswd`

Administer groups

Syntax:

```
gpasswd [OPTIONS] GROUP
```

Examples:

```
# Add user to group
sudo gpasswd -a kali docker

# Remove user from group
sudo gpasswd -d kali docker

# Set group administrators
sudo gpasswd -A kali developers

# Set group password (rarely used)
sudo gpasswd developers
```

Options:

Option	Description
`-a USER`	Add user to group
`-d USER`	Delete user from group
`-A USER`	Set group administrator
`-M USER,USER`	Set member list (replaces)

7.4 Switching Users & Sudo

Command: `su`

Switch User

Syntax:

```
su [OPTIONS] [USERNAME]
```

Examples:

```
# Switch to root (requires root password)
su
su -

# Switch to another user
su kali
su - kali     # Login shell (loads profile)

# Run single command as user
su -c "whoami" kali
```

`su` vs `su -`:

Command	Environment
`su user`	Keeps current environment
`su - user`	Full login shell (loads . bashrc, .profile)

Command: `sudo`

Execute command as superuser (or another user)

Syntax:

```
sudo [OPTIONS] COMMAND
```

Examples:

```
# Run as root
sudo apt update
sudo cat /etc/shadow

# Run as specific user
sudo -u postgres psql
sudo -u www-data touch /var/www/test

# Edit file with sudo
sudo nano /etc/hosts
sudoedit /etc/hosts       # Safer alternative

# Spawn root shell
sudo -s         # Shell with current environment
sudo -i         # Login shell (like su -)
sudo su -       # Another way

# List sudo privileges
sudo -l

# Validate/refresh sudo timestamp
sudo -v

# Remove sudo timestamp (require password again)
sudo -k
```

Common Options:

Option	Description
`-u USER`	Run as specified user
`-s`	Run shell
`-i`	Run login shell
`-l`	List allowed commands
`-v`	Validate (extend timeout)
`-k`	Invalidate timestamp
`-E`	Preserve environment

The Sudoers File

Location: `/etc/sudoers`

> ⚠ **NEVER edit directly! ** Use `visudo` — it validates syntax before saving.

```
sudo visudo
```

Sudoers Syntax:

```
user    host=(runas_user)    commands
```

Examples:

```
# Full root access
kali    ALL=(ALL: ALL) ALL

# Run specific commands without password
kali    ALL=(ALL) NOPASSWD:  /usr/bin/apt, /usr/bin/systemctl

# No password for all commands
kali    ALL=(ALL) NOPASSWD: ALL

# Group sudo access (note the %)
%sudo   ALL=(ALL: ALL) ALL
%admin  ALL=(ALL) NOPASSWD: ALL
```

Sudoers Tokens:

Token	Meaning
`ALL`	All hosts / all users / all commands
`NOPASSWD:`	Don't require password
`%groupname`	Apply to group
`!command`	Exclude command

Include Files:

```
# /etc/sudoers.d/ directory for custom rules
sudo visudo -f /etc/sudoers. d/custom_rules
```

Sudo Security for Pentesting

Check sudo rights (privilege escalation):

```
# What can current user run?
sudo -l

# Example output revealing escalation path:
# User kali may run the following commands:
#     (ALL) NOPASSWD: /usr/bin/vim
#     (ALL) NOPASSWD:  /usr/bin/find
```

Exploitable sudo entries:

```
# vim with sudo = root shell
sudo vim -c '! sh'

# find with sudo = root shell
sudo find /tmp -exec /bin/sh \;

# less with sudo = root shell
sudo less /etc/passwd
# Then type: ! sh
```

> **GTFOBins** lists sudo exploitation techniques for many binaries.

7.5 Password Management

Command: `passwd`

Change password

Syntax:

```
passwd [OPTIONS] [USERNAME]
```

Examples:

```
 # Change own password
passwd

# Change another user's password (root)
sudo passwd kali

# Lock account
sudo passwd -l username
# Adds !  prefix to password hash

# Unlock account
sudo passwd -u username

# Force password change on next login
sudo passwd -e username

# Set password expiry
sudo passwd -x 90 username    # Max 90 days
sudo passwd -n 7 username     # Min 7 days between changes

# View password status
sudo passwd -S username
# kali P 12/25/2024 0 99999 7 -1
# P=set, L=locked, NP=no password
```

Options:

Option	Description
`-l`	Lock account
`-u`	Unlock account
`-e`	Expire password (force change)
`-d`	Delete password (make empty)
`-S`	Show password status
`-x DAYS`	Maximum days before change required
`-n DAYS`	Minimum days between changes
`-w DAYS`	Warning days before expiry

Command: `chage`

Change password aging/expiry

Syntax:

```
chage [OPTIONS] USERNAME
```

Examples:

```
# View aging info
sudo chage -l kali

# Interactive mode
sudo chage kali

# Set expiry date
sudo chage -E 2025-12-31 kali

# Force change on next login
sudo chage -d 0 kali

# Set max days
sudo chage -M 90 kali
```

Setting Passwords Non-Interactively

```
# Using chpasswd (reads from stdin)
echo "username:newpassword" | sudo chpasswd

# Using openssl
sudo usermod -p $(openssl passwd -6 "password") username

# Using useradd with password
sudo useradd -m -p $(openssl passwd -6 "password") newuser
```

Key Terms

Term	Definition
UID	User ID — unique number identifying a user. Root = 0.
GID	Group ID — unique number identifying a group.
Primary Group	User's main group (from /etc/passwd). New files get this group.
Supplementary Groups	Additional groups user belongs to (from /etc/group).
Shadow File	`/etc/shadow` — stores encrypted passwords, readable only by root.
Sudoers	Configuration file controlling who can use sudo and how.
PAM	Pluggable Authentication Modules — framework for authentication.
GECOS	Comment field in /etc/passwd (historically: General Electric Comprehensive Operating System).

FAQs

Q: What's the difference between `useradd` and `adduser`?

> `useradd` is the low-level command — requires explicit options for home directory, shell, etc.
> `adduser` (Debian/Ubuntu) is a friendly wrapper — interactive, creates home, sets password.
> In scripts, use `useradd` for precise control.

Q: Why use `sudo` instead of logging in as root?

> - **Accountability:** sudo logs who ran what command
> - **Granularity:** Grant specific permissions, not full root
> - **Security:** No need to share root password

- **Accident prevention:** Commands require explicit `sudo` prefix

Q: I added a user to a group but it's not showing. Why?

> Group changes take effect on **next login**. Either:
>
> ```
> # Log out and back in, OR
> newgrp groupname    # Spawn new shell with group
> su - $USER          # Re-login as yourself
> ```

Q: How do I give a user sudo access?

> Add them to the `sudo` group:
>
> ```
> sudo usermod -aG sudo username
> ```
>
> Or add explicit entry in sudoers:
>
> ```
> sudo visudo
> # Add: username ALL=(ALL:ALL) ALL
> ```

Q: What's the difference between locking an account with `usermod -L` vs `passwd -l`?

> Both add `!` prefix to password hash, but:
>
> - `usermod -L` : Only locks password auth
> - `passwd -l` : Locks password auth
>
> Both still allow SSH key authentication! To fully disable:
>
> ```
> sudo usermod -s /usr/sbin/nologin username
> ```

Practice Tasks

Task 1: Create a Restricted Pentesting User

Objective: Create a user with limited privileges for safe testing

Steps:

```
# Create user with specific settings
sudo useradd -m -s /bin/bash -c "Pentest Practice User" pentester

# Set password
sudo passwd pentester

# Create restricted directory structure
sudo mkdir -p /home/pentester/{tools,targets,reports}
sudo chown -R pentester:pentester /home/pentester/

# Verify
id pentester
ls -la /home/pentester/

# Test login
su - pentester
whoami
pwd
exit
```

Optional — Add specific sudo permissions:

```
sudo visudo -f /etc/sudoers.d/pentester
# Add:  pentester ALL=(ALL) NOPASSWD:  /usr/bin/nmap, /usr/bin/nikto
```

Deliverable: Screenshot showing user creation and successful login

Task 2: Add Yourself to Sudo Group

Objective: Understand privilege escalation via group membership

Steps:

```
 # Check current groups
groups
id

# Add to sudo group
sudo usermod -aG sudo $USER

# Verify (won't show until re-login!)
groups   # Still old groups

# Re-login to apply
su - $USER
# OR
newgrp sudo

# Verify again
groups   # Now shows sudo

# Test sudo access
sudo whoami   # Should output:  root

# View sudo privileges
sudo -l
```

Deliverable: Before/after screenshot of `groups` command showing sudo added

---

Task 3: Lock and Unlock a User Account

Objective: Practice account security management

Steps:

```
 # Create test user
sudo useradd -m -s /bin/bash locktest
sudo passwd locktest

# Verify login works
su - locktest
exit

# Check account status
sudo passwd -S locktest
# locktest P 12/25/2024 0 99999 7 -1
#          ^-- P = Password set

# Lock the account
sudo passwd -l locktest
# OR
sudo usermod -L locktest

# Check status again
sudo passwd -S locktest
# locktest L 12/25/2024 0 99999 7 -1
#          ^-- L = Locked

# Verify shadow file
sudo grep locktest /etc/shadow
# locktest:!$y$j9T$.. .: ...
#          ^-- ! prefix = locked

# Try to login (should fail)
su - locktest
# su:  Authentication failure

# Unlock the account
sudo passwd -u locktest
# OR
sudo usermod -U locktest

# Verify unlocked
sudo passwd -S locktest
# locktest P ...

# Cleanup
sudo userdel -r locktest
```

Deliverable: Screenshots showing account status changes (P → L → P)

✅ Module 7 Checkpoint

Before moving to Module 8, confirm:

- [ ] Understand /etc/passwd and /etc/shadow structure
- [ ] Can create users with useradd and set appropriate options
- [ ] Can modify users with usermod (especially `-aG` for groups)
- [ ] Can create and manage groups
- [ ] Understand difference between `su` and `sudo`
- [ ] Can edit sudoers safely with visudo
- [ ] Can lock/unlock user accounts
- [ ] Know how to check sudo privileges for privilege escalation

---

Module 8: Disk & Storage Management

Introduction

Understanding disk and storage management is crucial for system administration, forensics, and incident response. You need to know how Linux organizes storage, how to mount external devices, analyze disk usage, and recover from filesystem errors.

This module covers partitions, filesystems, mounting, disk usage analysis, and swap — essential skills for managing Linux systems and handling forensic evidence.

8.1 Disk Partitions & Filesystems

Storage Hierarchy

```
Physical Disk (/dev/sda)
    │
    ├── Partition 1 (/dev/sda1) → Filesystem (ext4) → Mount Point (/boot)
    ├── Partition 2 (/dev/sda2) → Filesystem (ext4) → Mount Point (/)
    └── Partition 3 (/dev/sda3) → Filesystem (swap) → [Swap Space]
```

Device Naming

Device	Description
`/dev/sda`	First SATA/SCSI/USB disk
`/dev/sdb`	Second SATA/SCSI/USB disk
`/dev/sda1`	First partition on sda
`/dev/sda2`	Second partition on sda
`/dev/nvme0n1`	First NVMe SSD
`/dev/nvme0n1p1`	First partition on NVMe
`/dev/vda`	Virtual disk (VMs)
`/dev/mmcblk0`	SD card / eMMC
`/dev/sr0`	CD/DVD drive

View Disk Information

```
# List all block devices
lsblk
# NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
# sda      8:0    0   100G  0 disk
# ├─sda1   8:1    0   512M  0 part /boot
# ├─sda2   8:2    0    95G  0 part /
# └─sda3   8:3    0   4.5G  0 part [SWAP]

# Detailed disk info
lsblk -f
# Shows filesystem type, UUID, mount points

# List partitions
sudo fdisk -l
sudo fdisk -l /dev/sda

# Partition table details
sudo parted -l

# Disk UUIDs (for fstab)
blkid
sudo blkid /dev/sda1
```

Common Filesystems

Filesystem	Description	Use Case
ext4	Extended Filesystem 4	Default Linux filesystem
xfs	High-performance	Large files, servers
btrfs	B-tree FS	Snapshots, RAID
ntfs	Windows NT FS	Windows compatibility
vfat/fat32	FAT filesystem	USB drives, boot partitions
exfat	Extended FAT	Large files on USB
swap	Swap space	Virtual memory
tmpfs	Temporary FS	RAM-based, volatile
nfs	Network FS	Network file sharing

View Filesystem Information

```
# Filesystem type
df -T
# Filesystem     Type  1K-blocks    Used Available Use% Mounted on
# /dev/sda2      ext4   98765432 4567890  89012345   5% /

# Detailed filesystem info
sudo file -s /dev/sda1

# ext4 filesystem details
sudo dumpe2fs /dev/sda1 | head -30
sudo tune2fs -l /dev/sda1
```

8.2 Mounting & Unmounting

Mounting connects a filesystem to a directory (mount point) in the directory tree.

Command: `mount`

Mount a filesystem

Syntax:

```
mount [OPTIONS] DEVICE MOUNT_POINT
```

Examples:

```
 # Mount USB drive
sudo mount /dev/sdb1 /mnt/usb

# Mount with specific filesystem type
sudo mount -t ntfs /dev/sdb1 /mnt/windows

# Mount read-only
sudo mount -o ro /dev/sdb1 /mnt/evidence

# Mount with specific options
sudo mount -o rw,noexec,nosuid /dev/sdb1 /mnt/usb

# Mount ISO file
sudo mount -o loop image.iso /mnt/iso

# Mount network share (NFS)
sudo mount -t nfs server:/share /mnt/nfs

# Mount Windows share (CIFS/SMB)
sudo mount -t cifs //server/share /mnt/smb -o user=admin

# View all mounts
mount
mount | column -t
cat /proc/mounts
```

Common Mount Options:

Option	Description
`ro`	Read-only
`rw`	Read-write
`noexec`	Prevent execution of binaries
`nosuid`	Ignore SUID bits
`nodev`	Ignore device files
`loop`	Mount file as device (ISOs)
`uid=1000`	Set owner UID
`gid=1000`	Set group GID
`umask=022`	Set permission mask

Command: `umount`

Unmount a filesystem

Syntax:

```
umount [OPTIONS] MOUNT_POINT|DEVICE
```

Examples:

```
# Unmount by mount point
sudo umount /mnt/usb

# Unmount by device
sudo umount /dev/sdb1

# Force unmount (dangerous)
sudo umount -f /mnt/stuck

# Lazy unmount (detach now, cleanup when free)
sudo umount -l /mnt/busy
```

"Device is busy" Error:

```
# Find what's using the mount
lsof +D /mnt/usb
fuser -mv /mnt/usb

# Kill processes using it
fuser -k /mnt/usb

# Or move out of directory first
cd /
sudo umount /mnt/usb
```

`/etc/fstab` — Persistent Mounts

The fstab file defines filesystems to mount at boot.

```
cat /etc/fstab
```

Format:

```
device          mount_point    fs_type    options         dump  pass
/dev/sda1       /boot          ext4       defaults        0     2
/dev/sda2       /              ext4       errors=remount-ro 0   1
UUID=abc123...   /home          ext4       defaults        0     2
/dev/sda3       none           swap       sw              0     0
tmpfs           /tmp           tmpfs      defaults,noexec 0     0
```

Fields Explained:

Field	Description
`device`	Device name, UUID, or LABEL
`mount_point`	Directory to mount on
`fs_type`	Filesystem type
`options`	Mount options (comma-separated)
`dump`	Backup flag (0=no, 1=yes) — rarely used
`pass`	fsck order (0=skip, 1=root first, 2=others)

Best Practice — Use UUID:

```
# Get UUID
blkid /dev/sdb1
# /dev/sdb1: UUID="a1b2c3d4-..." TYPE="ext4"

# Use in fstab
UUID=a1b2c3d4-... /mnt/data ext4 defaults 0 2
```

Test fstab Without Reboot:

```
# Mount all from fstab
sudo mount -a

# Check for errors
sudo findmnt --verify
```

8.3 Disk Usage

Command: `df`

Disk Free — filesystem space usage

Syntax:

```
df [OPTIONS] [PATH]
```

Examples:

```
# Human-readable sizes
df -h
# Filesystem      Size  Used Avail Use% Mounted on
# /dev/sda2        94G  4. 5G   85G   5% /
# /dev/sda1       511M   50M  462M  10% /boot

# Show filesystem type
df -hT

# Specific filesystem
df -h /home
df -h .       # Current directory's filesystem

# Inodes instead of blocks
df -i

# Exclude filesystem types
df -h -x tmpfs -x devtmpfs
```

Key Metrics:

Column	Description
`Size`	Total filesystem size
`Used`	Space used
`Avail`	Space available
`Use%`	Percentage used
`Mounted on`	Mount point

> ⚠ **Alert Level**: Investigate when filesystem exceeds 80% usage

⚠️ **Alert Level:** Investigate when filesystem exceeds 80% usage.

Command: `du`

Disk Usage — estimate file/directory space

Syntax:

```
du [OPTIONS] [PATH]
```

Examples:

```
 # Summary of directory
du -sh /var/log
# 250M    /var/log

# All subdirectories
du -h /var/log

# Limit depth
du -h --max-depth=1 /var

# Sort by size (largest first)
du -sh /var/* 2>/dev/null | sort -rh | head -10

# Specific depth, sorted
du -h -d 1 /home | sort -rh

# Exclude pattern
du -sh --exclude='*.log' /var

# Apparent size vs disk usage
du -sh --apparent-size /var/log
```

Useful One-Liners:

```
 # Top 10 largest directories
du -h /var --max-depth=1 2>/dev/null | sort -rh | head -10

# Top 10 largest files
find /var -type f -exec du -h {} + 2>/dev/null | sort -rh | head -10

# Largest files over 100MB
find / -type f -size +100M -exec ls -lh {} \; 2>/dev/null
```

Command: `ncdu`

NCurses Disk Usage — interactive disk analyzer

```
 # Install
sudo apt install ncdu

# Run on directory
ncdu /var

# Scan and save for later
ncdu -o scan. json /
ncdu -f scan.json      # Load saved scan
```

ncdu Navigation:

Key	Action

Key	Action
↑/↓	Navigate
`Enter`	Enter directory
`d`	Delete selected
`n`	Sort by name
`s`	Sort by size
`g`	Show percentage graph
`q`	Quit

> ncdu is excellent for finding what's consuming disk space interactively!

8.4 Filesystem Checks

Command: `fsck`

Filesystem Check — repair filesystem errors

Syntax:

```
fsck [OPTIONS] DEVICE
```

> ⚠ **CRITICAL:** NEVER run fsck on mounted filesystems! Unmount first or boot to recovery mode.

Examples:

```
# Check filesystem (interactive)
sudo fsck /dev/sdb1

# Automatic repair
sudo fsck -y /dev/sdb1

# Check without fixing (safe)
sudo fsck -n /dev/sdb1

# Force check even if clean
sudo fsck -f /dev/sdb1

# Check specific filesystem type
sudo fsck. ext4 /dev/sdb1
```

Options:

Option	Description
`-y`	Auto-yes to all repairs
`-n`	No changes (read-only check)
`-f`	Force check even if marked clean
`-p`	Auto-repair (safe fixes only)
`-C`	Show progress

Force fsck on Next Boot:

```
# Create forcefsck file
sudo touch /forcefsck

# Or use tune2fs
sudo tune2fs -C 100 /dev/sda1    # Set mount count high
```

Disk Health Monitoring

```
# SMART status (requires smartmontools)
sudo apt install smartmontools
sudo smartctl -H /dev/sda         # Health status
sudo smartctl -a /dev/sda         # All SMART data

# Disk I/O stats
iostat -x 1

# Bad blocks check (SLOW)
sudo badblocks -v /dev/sdb
```

8.5 Swap Management

Swap is disk space used as virtual memory when RAM is full.

View Swap Status

```
# View swap usage
free -h
# total        used        free      shared  buff/cache   available
# Mem:           15Gi       3.2Gi       10Gi       256Mi       2.0Gi       11Gi
# Swap:         2.0Gi          0B       2.0Gi

# Detailed swap info
swapon --show
# NAME      TYPE      SIZE USED PRIO
# /dev/sda3 partition   2G   0B   -2

# View swap in /proc
cat /proc/swaps
```

Managing Swap

```
# Enable swap partition
sudo swapon /dev/sda3

# Disable swap
sudo swapoff /dev/sda3

# Disable all swap
sudo swapoff -a

# Enable all from fstab
sudo swapon -a
```

Creating Swap File

```
# Create 2GB swap file
sudo dd if=/dev/zero of=/swapfile bs=1M count=2048

# Or use fallocate (faster)
sudo fallocate -l 2G /swapfile

# Set permissions (IMPORTANT)
sudo chmod 600 /swapfile

# Format as swap
sudo mkswap /swapfile

# Enable
sudo swapon /swapfile

# Make permanent — add to fstab
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab

# Verify
swapon --show
```

Swap Priority and Swappiness

```
# View swappiness (0-100, lower = less swap use)
cat /proc/sys/vm/swappiness
# Default: 60

# Temporary change
sudo sysctl vm. swappiness=10

# Permanent change
echo 'vm.swappiness=10' | sudo tee -a /etc/sysctl.conf

# Set swap priority
sudo swapon -p 100 /swapfile    # Higher priority used first
```

Key Terms

Term	Definition
Partition	Logical division of a physical disk.
Filesystem	Structure organizing data on a partition (ext4, NTFS, XFS).
Mount Point	Directory where a filesystem is attached to the directory tree.
UUID	Universally Unique Identifier — unique ID for a filesystem.
Inode	Data structure storing file metadata. Finite per filesystem.
Block	Smallest unit of storage on disk (usually 4KB).
Swap	Disk space used as virtual memory extension.
fstab	File System Table — defines mounts at boot time.
fsck	Filesystem check utility for repairs.

❓ FAQs

Q: What's the difference between `df` and `du`?

> - `df` shows **filesystem** space usage (total, used, available)
> - `du` shows **directory/file** space usage (what's consuming space)
>
> Use `df` to see if a filesystem is full, `du` to find what's using the space.

Q: Why does `df` show different "available" than total minus used?

> Linux reserves ~5% of ext filesystems for root to prevent complete fill. This prevents system lockup if disk fills. Reduce with:
>
> ```
> sudo tune2fs -m 1 /dev/sda1    # Reserve only 1%
> ```

Q: Why can't I unmount a filesystem?

> Something is using it. Find what:
>
> ```
> lsof +D /mnt/usb     # Open files
> fuser -mv /mnt/usb   # Processes using mount
> ```
>
> Then either close those programs or:
>
> ```
> sudo umount -l /mnt/usb    # Lazy unmount
> ```

Q: Should I use UUID or device name in fstab?

> **Always use UUID. ** Device names (`/dev/sdb1`) can change if you add/remove drives or change port order. UUIDs are permanent:
>
> ```
> UUID=abc123...  /mnt/data ext4 defaults 0 2
> ```

Q: How much swap do I need?

> General guidelines:
>
> | RAM | Swap (No Hibernate) | Swap (With Hibernate) |
> |---|---|---|
> | ≤2GB | 2x RAM | 3x RAM |
> | 2-8GB | Equal to RAM | 2x RAM |
> | 8-64GB | 4-8GB | RAM + 2GB |
> | >64GB | 4GB minimum | Not recommended |
>
> For 16GB RAM (your system): 4-8GB swap is sufficient.

🎯 Practice Tasks

Task 1: Check Disk Usage — Find Space Hogs

Objective: Identify largest directories consuming space

Steps:

```
# Overall filesystem usage
df -h

# Find largest directories in /var
sudo du -h --max-depth=1 /var 2>/dev/null | sort -rh | head -10

# Find largest files system-wide
sudo find / -type f -size +50M -exec ls -lh {} \; 2>/dev/null | sort -k5 -rh | head -10

# Interactive analysis with ncdu
sudo apt install ncdu
sudo ncdu /var

# Log files specifically (often the culprit)
sudo du -sh /var/log/*  2>/dev/null | sort -rh | head -5
```

Deliverable:

- List of top 5 largest directories
- Screenshot of ncdu interface

Task 2: Mount USB Drive — Forensic Evidence Collection

Objective: Practice mounting external media read-only

Steps:

```
# Identify USB drive (before and after inserting)
lsblk

# After inserting USB, find new device
lsblk
# Should see new device like /dev/sdb or /dev/sdc

# Get filesystem info
sudo blkid /dev/sdb1

# Create mount point
sudo mkdir -p /mnt/evidence

# Mount READ-ONLY (forensic best practice)
sudo mount -o ro,noexec,nosuid /dev/sdb1 /mnt/evidence

# Verify read-only mount
mount | grep evidence
# Should show:  /dev/sdb1 on /mnt/evidence type vfat (ro,noexec,nosuid)

# Copy evidence (example)
cp -r /mnt/evidence/* ~/case_001/

# Calculate hashes (forensic integrity)
find /mnt/evidence -type f -exec md5sum {} \; > ~/case_001/hashes.md5

# Unmount when done
cd /
sudo umount /mnt/evidence
```

Deliverable:

- Screenshot of read-only mount
- Hash file of evidence

Task 3: Analyze /etc/fstab

Objective: Understand persistent mount configuration

Steps:

```
# View fstab
cat /etc/fstab

# Document each line
```

Sample fstab Analysis:

```
# /etc/fstab — Filesystem Table

# Line 1: Root filesystem
UUID=abc123... / ext4 errors=remount-ro 0 1
# - Uses UUID for device identification
# - Mounted at root (/)
# - ext4 filesystem
# - If errors occur, remount read-only
# - 0 = don't dump
# - 1 = check first (root partition)

# Line 2: Boot partition
UUID=def456... /boot ext4 defaults 0 2
# - Separate boot partition
# - defaults = rw,suid,dev,exec,auto,nouser,async
# - 2 = check after root

# Line 3: Swap
UUID=789xyz... none swap sw 0 0
# - Swap partition (no mount point)
# - sw = swap options
# - 0 0 = no dump, no fsck

# Line 4: Temporary filesystem
tmpfs /tmp tmpfs defaults,noexec,nosuid 0 0
# - RAM-based filesystem
# - noexec = can't execute binaries (security)
# - nosuid = SUID bits ignored (security)
```

Verify fstab:

```
# Check for syntax errors
sudo findmnt --verify

# Test mount all
sudo mount -a
echo $?   # 0 = success
```

Deliverable:

- Your annotated fstab with explanations for each line
- Output of `findmnt --verify`

✅ Module 8 Checkpoint

Before moving to Module 9, confirm:

- ☐ Understand partition and filesystem concepts
- ☐ Can view disk information with lsblk, fdisk, blkid
- ☐ Can mount and unmount filesystems
- ☐ Know how to use mount options (ro, noexec, nosuid)
- ☐ Understand /etc/fstab structure and UUID usage
- ☐ Can analyze disk usage with df, du, ncdu
- ☐ Know when and how to use fsck
- ☐ Can manage swap space

Module 9: Package Management

Introduction

Package management is how Linux systems install, update, and remove software. Unlike Windows where you download `.exe` files from random websites, Linux uses **repositories** — trusted sources of pre-compiled software managed by your distribution.

Understanding package management is essential for setting up pentesting tools, maintaining system security, and troubleshooting dependency issues. This module covers Debian/Kali's APT system (your primary focus), plus alternatives for broader Linux knowledge.

9.1 What are Packages?

A **package** is a bundled archive containing:

- Compiled program binaries
- Configuration files
- Documentation
- Dependency information
- Installation scripts

Package Formats

Format	Extension	Distributions
DEB	`.deb`	Debian, Ubuntu, Kali, Mint
RPM	`.rpm`	Fedora, RHEL, CentOS, openSUSE
Pacman	`.pkg. tar.zst`	Arch, Manjaro
Source	`.tar.gz`	Universal (compile yourself)

Package Management Layers

```
┌─────────────────────────────────────────────┐
│              High-Level Tools               │
│         apt, apt-get, aptitude              │
│    (Handle dependencies, repositories)      │
├─────────────────────────────────────────────┤
│              Low-Level Tools                │
│                   dpkg                      │
│     (Install/remove individual packages)    │
├─────────────────────────────────────────────┤
│              Package Files                  │
│                 . deb files                 │
└─────────────────────────────────────────────┘
```

9.2 APT (Debian/Kali/Ubuntu)

APT (Advanced Package Tool) is the primary package manager for Debian-based systems.

`apt` vs `apt-get`

Feature	`apt`	`apt-get`
Purpose	End-user friendly	Scripting, backward

Feature	`apt`	compatibility `apt-get`
Progress bar	✅ Yes	❌ No
Colored output	✅ Yes	❌ No
Recommended for	Interactive use	Scripts, automation

> 💡 **Use `apt` for daily work, `apt-get` in scripts for stability.**

Command: `apt update`

Refresh package lists from repositories

```
sudo apt update
```

What it does:

- Downloads latest package lists from configured repos
- Checks for available updates
- Does NOT install anything

> ⚠ **Always run before installing or upgrading! **

Command: `apt upgrade`

Upgrade installed packages

```
# Standard upgrade (safe)
sudo apt upgrade

# Full upgrade (may remove packages if needed)
sudo apt full-upgrade

# Simulate upgrade (see what would happen)
apt upgrade --simulate
apt upgrade -s
```

Difference:

Command	Behavior
`upgrade`	Never removes packages
`full-upgrade`	May remove packages to resolve conflicts
`dist-upgrade`	Same as full-upgrade (older syntax)

Command: `apt install`

Install packages

```
# Install single package
sudo apt install nmap

# Install multiple packages
sudo apt install nmap nikto gobuster

# Install specific version
sudo apt install nmap=7.93-1

# Install without prompting
sudo apt install -y nmap

# Fix broken dependencies during install
sudo apt install -f

# Install . deb file (uses apt for dependencies)
sudo apt install ./package.deb

# Install recommended packages too
sudo apt install --install-recommends package

# Install only suggested packages
sudo apt install --install-suggests package

# Reinstall package
sudo apt install --reinstall nmap
```

Useful Options:

Option	Description
`-y`	Assume yes to prompts
`-f`	Fix broken dependencies
`--no-install-recommends`	Skip recommended packages
`--reinstall`	Reinstall package
`-s`	Simulate (dry run)

Command: `apt remove` / `apt purge`

Remove packages

```
# Remove package (keep config files)
sudo apt remove nmap

# Remove package AND config files
sudo apt purge nmap

# Remove with dependencies that are no longer needed
sudo apt remove --autoremove nmap

# Remove unused dependencies
sudo apt autoremove
```

Difference:

Command	Binaries	Config Files	Dependencies

Command	Binaries	Config Files	Dependencies
`remove`	✅ Removed	❌ Kept	❌ Kept
`purge`	✅ Removed	✅ Removed	❌ Kept
`autoremove`	—	—	✅ Removed

Command: `apt search` / `apt show`

Find and inspect packages

```
# Search for package
apt search nmap
apt search "network scanner"

# Show package details
apt show nmap

# List all versions available
apt list -a nmap

# List installed packages
apt list --installed

# List upgradable packages
apt list --upgradable

# Check if package is installed
apt list --installed | grep nmap
dpkg -l | grep nmap
```

Command: `apt-cache`

Query package cache

```
# Search packages
apt-cache search nmap

# Show package info
apt-cache show nmap

# Show dependencies
apt-cache depends nmap

# Show reverse dependencies (what depends on it)
apt-cache rdepends nmap

# Show package statistics
apt-cache stats

# Show policy (versions, priorities)
apt-cache policy nmap
```

Common APT Operations

```
# Full system update routine
sudo apt update && sudo apt upgrade -y

# Clean up
sudo apt autoremove          # Remove unused dependencies
sudo apt autoclean           # Remove old package files
sudo apt clean               # Remove all cached packages

# Check for issues
sudo apt --fix-broken install
sudo dpkg --configure -a     # Configure pending packages

# Hold package (prevent upgrades)
sudo apt-mark hold package_name

# Unhold package
sudo apt-mark unhold package_name

# View held packages
apt-mark showhold
```

9.3 DPKG (Low-Level)

dpkg handles individual `.deb` files directly — no dependency resolution.

Command: `dpkg`

Debian Package manager

```
# Install . deb file
sudo dpkg -i package.deb

# Remove package
sudo dpkg -r package_name

# Remove with config files
sudo dpkg -P package_name

# List installed packages
dpkg -l
dpkg -l | grep nmap

# List files in installed package
dpkg -L nmap

# Find which package owns a file
dpkg -S /usr/bin/nmap

# Show package info
dpkg -s nmap

# Extract without installing
dpkg -x package. deb /tmp/extracted/

# Reconfigure package
sudo dpkg-reconfigure package_name
```

Common dpkg Options:

Option	Description
`-i`	Install package

Option	Description
	Remove package
`-P`	Purge (remove with configs)
`-l`	List packages
`-L`	List files in package
`-S`	Search for file owner
`-s`	Show package status

Fixing Dependency Issues:

```
# After dpkg -i fails with dependencies
sudo apt install -f
# This installs missing dependencies
```

9.4 YUM/DNF (Fedora/RHEL)

For reference when working with Red Hat-based systems:

DNF (Modern - Fedora 22+, RHEL 8+)

```
# Update
sudo dnf update

# Install
sudo dnf install nmap

# Remove
sudo dnf remove nmap

# Search
dnf search nmap

# Info
dnf info nmap

# List installed
dnf list installed
```

YUM (Legacy - RHEL 7, CentOS 7)

```
sudo yum update
sudo yum install nmap
sudo yum remove nmap
yum search nmap
```

9.5 Managing Repositories

APT Repository Configuration

Main configuration: `/etc/apt/sources.list`
Additional repos: `/etc/apt/sources.list. d/*. list`

```
# View current sources
cat /etc/apt/sources.list
ls /etc/apt/sources.list. d/
```

Sources. list Format

```
deb http://http.kali.org/kali kali-rolling main contrib non-free non-free-firmware
│   │                        │            │
│   │                        │            └── Components (sections)
│   │                        └── Distribution/codename
│   └── Repository URL
└── Type (deb = binary, deb-src = source)
```

Components:

Component	Description
`main`	Officially supported free software
`contrib`	Free software with non-free dependencies
`non-free`	Non-free software
`non-free-firmware`	Firmware packages

Adding Repositories

```
# Method 1: Add PPA (Ubuntu)
sudo add-apt-repository ppa: user/repo
sudo apt update

# Method 2: Add custom repo manually
echo "deb http://example.com/repo stable main" | sudo tee /etc/apt/sources.list. d/custom.list

# Method 3: Add GPG key for repo
wget -qO - https://example.com/key.gpg | sudo apt-key add -
# Or modern method:
wget -qO - https://example.com/key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/example.gpg
echo "deb [signed-by=/usr/share/keyrings/example.gpg] https://example.com/repo stable main" | sudo tee /etc/apt/sources. list.d/exam

# Update after adding
sudo apt update
```

Remove Repository

```
# Remove PPA
sudo add-apt-repository --remove ppa:user/repo

# Or manually delete file
sudo rm /etc/apt/sources. list.d/custom.list
sudo apt update
```

9.6 Installing from Source

When a tool isn't in repositories, compile from source.

Standard Compilation Process

```
# 1. Install build dependencies
sudo apt install build-essential git

# 2. Clone repository
git clone https://github.com/user/tool.git
cd tool

# 3. Check for build instructions
cat README.md
cat INSTALL

# 4. Standard build process
./configure          # Check system, generate Makefile
make                 # Compile source code
sudo make install    # Install binaries to /usr/local/

# Alternative: If using CMake
mkdir build && cd build
cmake ..
make
sudo make install
```

Example: Installing Tool from GitHub

```
# Example: Installing a Go-based tool
# First, install Go if needed
sudo apt install golang-go

# Clone and build
git clone https://github.com/projectdiscovery/nuclei.git
cd nuclei/v2/cmd/nuclei
go build
sudo mv nuclei /usr/local/bin/

# Verify
nuclei -version
```

Python Tools

```
# Clone repo
git clone https://github.com/user/python-tool.git
cd python-tool

# Method 1: pip install
pip install .
# Or
pip install -r requirements.txt
python setup.py install

# Method 2: pipx (isolated environments)
pipx install .

# Method 3: Run directly
python3 tool.py
```

Uninstalling Source-Compiled Software

```
# If Makefile supports it
sudo make uninstall

# Otherwise, manually remove
sudo rm /usr/local/bin/toolname
```

9.7 Snap & Flatpak

Modern universal package formats that bundle dependencies.

Snap

```
# Install snapd (if not installed)
sudo apt install snapd

# Search
snap find nmap

# Install
sudo snap install package

# List installed
snap list

# Update
sudo snap refresh

# Remove
sudo snap remove package

# Info
snap info package
```

Snap Characteristics:

- Sandboxed applications
- Automatic updates
- Larger package sizes
- Managed by Canonical (Ubuntu)

Flatpak

```
# Install flatpak
sudo apt install flatpak

# Add Flathub repository
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo

# Search
flatpak search appname

# Install
flatpak install flathub org.app. Name

# Run
flatpak run org.app.Name

# Update all
flatpak update

# Remove
flatpak uninstall org.app.Name
```

> **For pentesting:** Stick with APT and native packages. Snap/Flatpak sandboxing can interfere with security tools.

Key Terms

Term	Definition
Package	Bundled software archive with binaries, configs, and metadata.
Repository	Server hosting packages for a distribution.
Dependency	Package required by another package to function.
APT	Advanced Package Tool — high-level package manager for Debian.
dpkg	Low-level Debian package manager — handles individual .deb files.
Sources.list	Configuration file listing package repositories.
PPA	Personal Package Archive — third-party Ubuntu repositories.
Cache	Local storage of downloaded packages (`/var/cache/apt/archives/`).

FAQs

Q: What's the difference between `apt` and `apt-get`?

> `apt` is the modern, user-friendly tool with progress bars and colors. `apt-get` is the traditional tool, preferred in scripts for backward compatibility. For interactive use, prefer `apt`.

Q: Why does `apt update` not install updates?

> `apt update` only refreshes the package list (metadata). It tells APT what's available. `apt upgrade` actually downloads and installs the updates. Always run both:
>
> ```
> sudo apt update && sudo apt upgrade
> ```

Q: How do I install a .deb file with dependencies?

> Use `apt` instead of `dpkg`:
>
> ```
> sudo apt install ./package.deb
> ```
>
> This resolves and installs dependencies automatically.

Q: How do I find which package provides a command?

> ```
> apt-file search /usr/bin/command    # Need apt-file installed
> # OR
> dpkg -S /usr/bin/command            # For installed packages
> # OR
> apt search command-name
> ```

Q: How do I downgrade a package to an older version?

> ```
> # List available versions
> apt list -a package_name
>
> # Install specific version
> sudo apt install package_name=version
>
> # Hold to prevent upgrade
> sudo apt-mark hold package_name
> ```

Practice Tasks

Task 1: Install a Pentesting Tool from Kali Repos

Objective: Practice APT package installation

Steps:

```
 # Update package lists
sudo apt update

# Search for a tool (e.g., gobuster)
apt search gobuster

# View package details
apt show gobuster

# Install the tool
sudo apt install -y gobuster

# Verify installation
gobuster version
which gobuster

# View installed files
dpkg -L gobuster

# Check dependencies
apt-cache depends gobuster
```

Alternative Tools to Try:

- `feroxbuster` — Fast content discovery
- `subfinder` — Subdomain enumeration
- `httpx` — HTTP probing tool

Deliverable: Screenshot of tool running with `--help` or `version`

Task 2: Install Tool from GitHub (Compile from Source)

Objective: Practice building software from source

Example: Installing ffuf (Fast web fuzzer)

```
# Install Go (if not installed)
sudo apt install golang-go

# Set Go path
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin

# Method 1: Using go install
go install github. com/ffuf/ffuf/v2@latest

# Verify
ffuf -V

# Method 2: Clone and build manually
git clone https://github.com/ffuf/ffuf. git
cd ffuf
go build
sudo mv ffuf /usr/local/bin/

# Verify
which ffuf
ffuf -h
```

Alternative — Python Tool (sqlmap update):

```
# Clone latest version
git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git ~/tools/sqlmap

# Run
python3 ~/tools/sqlmap/sqlmap.py --version

# Create alias
echo "alias sqlmap='python3 ~/tools/sqlmap/sqlmap.py'" >> ~/.bashrc
source ~/.bashrc
```

Deliverable: Screenshot showing tool installed from source running successfully

---

Task 3: List Installed Packages and Find Outdated Ones

Objective: Practice package auditing

Steps:

```
# List all installed packages
dpkg -l | wc -l           # Count
dpkg -l > installed_packages.txt

# List with apt
apt list --installed

# Find upgradable packages
sudo apt update
apt list --upgradable

# Show only package names that need updates
apt list --upgradable 2>/dev/null | grep -v "Listing" | cut -d/ -f1

# Detailed upgrade info
apt-get -s upgrade | grep "^Inst"

# Check specific package version
apt-cache policy nmap

# Find manually installed packages (not auto-installed as dependencies)
apt-mark showmanual | head -20

# Compare with available versions
for pkg in nmap nikto gobuster; do
    echo "=== $pkg ==="
    apt-cache policy $pkg | head -4
done
```

Security Audit:

```
# Check for packages with known vulnerabilities (requires debsecan)
sudo apt install debsecan
debsecan --suite kali-rolling
```

Deliverable:

- List of outdated packages needing upgrade
- Count of total installed packages

Module 9 Checkpoint

Before moving to Module 10, confirm:

- ☐ Can update and upgrade system with apt
- ☐ Can install, remove, and purge packages
- ☐ Understand difference between apt and dpkg
- ☐ Can search for and get info about packages
- ☐ Know how to add and manage repositories
- ☐ Can install software from source (GitHub)
- ☐ Can list and audit installed packages

Module 10: Networking Basics

Introduction

Networking is the **heart of cybersecurity**. Every attack, every defense, every piece of reconnaissance happens over a network. Understanding Linux networking commands is mandatory for penetration testing, network analysis, and system administration.

This module covers network configuration, connectivity testing, DNS enumeration, port analysis, SSH, and file transfers — the essential toolkit for any security professional.

10.1 Network Interface Commands

`ifconfig` VS `ip`

Feature	`ifconfig` (legacy)	`ip` (modern)
Package	`net-tools`	`iproute2` (default)
Status	Deprecated	Current standard
Features	Basic	Advanced (namespaces, etc.)

> 💡 Learn `ip` — it's the future. Know `ifconfig` — you'll see it everywhere.

Command: `ip`

Modern network configuration tool

View Interfaces:

```
# Show all interfaces
ip addr
ip a              # Short form

# Show specific interface
ip addr show eth0

# Show only IPv4
ip -4 addr

# Show only IPv6
ip -6 addr

# Brief format
ip -br addr
# lo               UNKNOWN        127.0.0.1/8
# eth0             UP             192.168.1.100/24
```

Manage Interfaces:

```
# Bring interface up
sudo ip link set eth0 up

# Bring interface down
sudo ip link set eth0 down

# Set IP address
sudo ip addr add 192.168.1.100/24 dev eth0

# Remove IP address
sudo ip addr del 192.168.1.100/24 dev eth0

# Set MAC address
sudo ip link set eth0 down
sudo ip link set eth0 address 00:11:22:33:44:55
sudo ip link set eth0 up
```

Routing:

```
# Show routing table
ip route
ip r

# Show specific route
ip route get 8.8.8.8

# Add default gateway
sudo ip route add default via 192.168.1.1

# Add specific route
sudo ip route add 10.0.0.0/8 via 192.168.1.1

# Delete route
sudo ip route del 10.0.0.0/8
```

Neighbors (ARP):

```
# Show ARP cache
ip neigh
ip n

# Flush ARP cache
sudo ip neigh flush all
```

Command: `ifconfig` (Legacy)

Interface configuration

```
# Install if missing
sudo apt install net-tools

# Show all interfaces
ifconfig
ifconfig -a        # Include down interfaces

# Show specific interface
ifconfig eth0

# Set IP address
sudo ifconfig eth0 192.168.1.100 netmask 255.255.255.0

# Bring up/down
sudo ifconfig eth0 up
sudo ifconfig eth0 down
```

Other Network Info Commands

```
# View all network info
nmcli device show
nmcli connection show

# Network hardware
lshw -class network
lspci | grep -i net
lsusb | grep -i net

# Wireless interfaces
iwconfig
iw dev

# Hostname
hostname
hostname -I          # IP address
cat /etc/hostname
```

10.2 Network Configuration Files

File	Purpose
`/etc/hostname`	System hostname
`/etc/hosts`	Static hostname to IP mapping
`/etc/resolv.conf`	DNS resolver configuration
`/etc/network/interfaces`	Debian network config (legacy)
`/etc/netplan/*. yaml`	Ubuntu network config (modern)
`/etc/NetworkManager/`	NetworkManager configuration

DNS Configuration

```
# View current DNS
cat /etc/resolv.conf

# Modify DNS (temporary)
echo "nameserver 8.8.8.8" | sudo tee /etc/resolv. conf

# Modern systems:  NetworkManager manages resolv.conf
# Edit connections instead:
nmcli con mod "Wired connection 1" ipv4.dns "8.8.8.8 8.8.4.4"
nmcli con up "Wired connection 1"
```

Hosts File

```
cat /etc/hosts
# 127.0.0.1   localhost
# 192.168.1.50  target.local
# 10.10.10.10   victim.htb

# Add entry
echo "10.10.10.10  box.htb" | sudo tee -a /etc/hosts
```

10.3 Checking Connectivity

Command: `ping`

Test reachability via ICMP

Syntax:

```
ping [OPTIONS] HOST
```

Examples:

```
 # Basic ping (runs until Ctrl+C)
ping google.com

# Limit count
ping -c 4 google.com

# Continuous with timestamp
ping -D google.com

# Set interval
ping -i 0.5 google.com      # Every 0.5 seconds

# Flood ping (root, stress test)
sudo ping -f google.com

# Set packet size
ping -s 1000 google.com     # 1000 bytes

# Quiet mode (summary only)
ping -c 5 -q google. com

# Set timeout
ping -W 2 google.com        # 2 second timeout

# Use specific interface
ping -I eth0 google. com
```

Command: `traceroute`

Trace packet path to destination

Syntax:

```
traceroute [OPTIONS] HOST
```

Examples:

```
# Basic traceroute (ICMP)
traceroute google.com

# Use TCP (bypass firewalls blocking ICMP)
sudo traceroute -T -p 443 google.com

# Use UDP
traceroute -U google.com

# Maximum hops
traceroute -m 20 google.com

# Don't resolve hostnames (faster)
traceroute -n google.com

# Set timeout
traceroute -w 3 google. com
```

Command: `mtr`

Combined ping + traceroute (real-time)

```
# Install
sudo apt install mtr

# Interactive mode
mtr google.com

# Report mode (non-interactive)
mtr -r -c 10 google.com

# No DNS resolution
mtr -n google.com

# TCP mode
mtr --tcp -P 443 google.com
```

MTR Output Explained:

Column	Description
`Loss%`	Packet loss percentage
`Snt`	Packets sent
`Last`	Last RTT
`Avg`	Average RTT
`Best`	Minimum RTT
`Wrst`	Maximum RTT
`StDev`	Standard deviation

10.4 DNS Tools

Command: `nslookup`

Query DNS servers

Syntax:

```
nslookup [OPTIONS] HOST [SERVER]
```

Examples:

```
# Basic lookup
nslookup google.com

# Use specific DNS server
nslookup google.com 8.8.8.8

# Reverse lookup (IP to hostname)
nslookup 142.250.185.78

# Interactive mode
nslookup
> set type=MX
> google.com
> exit
```

Command: `dig`

DNS lookup utility (more powerful than nslookup)

Syntax:

```
dig [@SERVER] NAME [TYPE]
```

Examples:

```
# Basic A record lookup
dig google.com

# Specific record types
dig google.com A        # IPv4 address
dig google.com AAAA     # IPv6 address
dig google.com MX       # Mail servers
dig google.com NS       # Name servers
dig google.com TXT      # TXT records
dig google.com ANY      # All records (often blocked)
dig google.com SOA      # Start of Authority

# Use specific DNS server
dig @8.8.8.8 google.com

# Short output
dig +short google.com

# Trace delegation
dig +trace google.com

# Reverse lookup
dig -x 142.250.185.78

# AXFR zone transfer (if allowed)
dig axfr @ns1.target.com target.com

# Show all details
dig google.com +noall +answer

# Batch lookup
dig -f domains.txt +short
```

dig Output Sections:

Section	Content
QUESTION	What was queried
ANSWER	Response records
AUTHORITY	Authoritative nameservers
ADDITIONAL	Extra helpful records

Command: `host`

Simple DNS lookup

```
 # Basic lookup
host google.com

# Specific type
host -t MX google.com
host -t NS google.com

# Reverse lookup
host 142.250.185.78

# Use specific server
host google.com 8.8.8.8

# Verbose
host -v google. com
```

DNS Recon Commands

```
 # Find all subdomains (zone transfer attempt)
dig axfr @ns1.target.com target.com

# Brute force subdomains
for sub in www mail ftp admin; do
    host $sub. target.com
done

# Using dnsrecon tool
dnsrecon -d target.com

# Using fierce
fierce --domain target.com

# Using subfinder
subfinder -d target.com
```

10.5 Ports & Connections

Command: `netstat` (Legacy)

Network statistics

```
# Install (if needed)
sudo apt install net-tools

# All connections
netstat -a

# Listening ports only
netstat -l

# TCP connections
netstat -t

# UDP connections
netstat -u

# Show numeric addresses (skip DNS resolution)
netstat -n

# Show process using port
netstat -p

# Common combination:  listening TCP/UDP with PIDs
sudo netstat -tulpn

# Show routing table
netstat -r

# Show interface statistics
netstat -i

# Continuous monitoring
netstat -c
```

Command: `ss`

Socket Statistics (modern replacement for netstat)

Syntax:

```
ss [OPTIONS]
```

Examples:

```
# All connections
ss -a

# Listening sockets
ss -l

# TCP sockets
ss -t

# UDP sockets
ss -u

# Show processes
ss -p

# Numeric output
ss -n

# Common combination
sudo ss -tulpn
# -t TCP
# -u UDP
# -l listening
# -p processes
# -n numeric

# Filter by state
ss -t state established
ss -t state listening

# Filter by port
ss -tn dport = : 22
ss -tn sport = :80
ss -tn '( dport = : 443 or dport = :80 )'

# Show memory usage
ss -m

# Show timer info
ss -o

# Extended info
ss -e

# IPv4 only
ss -4

# IPv6 only
ss -6
```

ss States:

State	Description
`ESTABLISHED`	Active connection
`SYN-SENT`	Waiting for connection
`SYN-RECV`	Connection request received
`LISTEN`	Waiting for connections
`TIME-WAIT`	Connection closing

State	Description
`CLOSE-WAIT`	Remote closed, waiting local
`CLOSED`	Not in use

Command: `lsof`

List open files (including network sockets)

```
# Install if needed
sudo apt install lsof

# All network connections
sudo lsof -i

# Specific port
sudo lsof -i : 80
sudo lsof -i :22

# Specific protocol and port
sudo lsof -i tcp:443
sudo lsof -i udp: 53

# Specific host
sudo lsof -i @192.168.1.1

# Files opened by process
lsof -p 1234

# Files opened by user
lsof -u kali

# Listening ports
sudo lsof -i -P -n | grep LISTEN

# Who is using a file
lsof /var/log/syslog
```

10.6 SSH Basics

SSH (Secure Shell) — encrypted remote access protocol.

Connecting

```
# Basic connection
ssh user@hostname
ssh user@192.168.1.100

# Specify port
ssh -p 2222 user@host

# Verbose (debugging)
ssh -v user@host
ssh -vvv user@host      # Maximum verbosity

# Force password authentication
ssh -o PreferredAuthentications=password user@host

# Use specific key
ssh -i ~/.ssh/id_rsa user@host

# Execute single command
ssh user@host "uname -a"
ssh user@host 'cat /etc/passwd'

# Forward local port (local port forwarding)
ssh -L 8080:localhost:80 user@host
# Access remote's port 80 via local port 8080

# Forward remote port (reverse tunnel)
ssh -R 9090:localhost:22 user@host
# Remote can access your SSH via their port 9090

# Dynamic port forwarding (SOCKS proxy)
ssh -D 1080 user@host
# Configure browser to use localhost:1080 as SOCKS proxy

# Keep connection alive
ssh -o ServerAliveInterval=60 user@host

# Disable host key checking (insecure, for CTF/testing)
ssh -o StrictHostKeyChecking=no user@host
```

SSH Key Management

```
# Generate key pair
ssh-keygen -t ed25519 -C "kali@mybox"
# OR RSA (legacy compatible)
ssh-keygen -t rsa -b 4096

# Key locations
ls -la ~/.ssh/
# id_ed25519      <- Private key (NEVER share)
# id_ed25519.pub  <- Public key (share freely)

# Copy public key to remote
ssh-copy-id user@host
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@host

# Manual copy
cat ~/.ssh/id_ed25519.pub | ssh user@host "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"

# Set correct permissions
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_ed25519
chmod 644 ~/.ssh/id_ed25519.pub
chmod 600 ~/.ssh/authorized_keys

# Start SSH agent (key caching)
eval $(ssh-agent)
ssh-add ~/.ssh/id_ed25519

# List added keys
ssh-add -l
```

SSH Config File

Location: `~/.ssh/config`

```
# Example config
Host target
    HostName 192.168.1.100
    User admin
    Port 2222
    IdentityFile ~/.ssh/target_key

Host *.internal
    User root
    ProxyJump jumphost

Host jumphost
    HostName jump.company.com
    User kali
    IdentityFile ~/.ssh/jump_key
```

Usage:

```
ssh target          # Uses config settings
ssh web.internal    # Jumps through jumphost
```

10.7 File Transfer

Command: `scp`

Secure Copy — transfer files over SSH

Syntax:

```
scp [OPTIONS] SOURCE DESTINATION
```

Examples:

```
# Local to remote
scp file.txt user@host:/path/
scp -r folder/ user@host:/path/

# Remote to local
scp user@host:/path/file.txt ./
scp -r user@host:/path/folder/ ./

# Remote to remote
scp user1@host1:/file user2@host2:/path/

# Specific port
scp -P 2222 file.txt user@host:/path/

# Preserve attributes
scp -p file.txt user@host:/path/

# Use specific key
scp -i ~/.ssh/key file.txt user@host:/path/

# Bandwidth limit
scp -l 1000 file.txt user@host:/path/  # KB/s

# Compression
scp -C largefile.tar user@host:/path/
```

Command: `rsync`

Remote sync — efficient file synchronization

Syntax:

```
rsync [OPTIONS] SOURCE DESTINATION
```

Examples:

```
# Local sync
rsync -av source/ dest/

# Local to remote
rsync -av folder/ user@host:/path/

# Remote to local
rsync -av user@host:/path/ ./folder/

# Common options
rsync -avz folder/ user@host:/path/    # Compressed
rsync -avP folder/ user@host:/path/    # Progress + partial

# Delete files not in source (MIRROR)
rsync -av --delete source/ dest/

# Exclude files
rsync -av --exclude='*.log' source/ dest/

# Dry run (preview)
rsync -avn source/ dest/

# Over specific SSH port
rsync -av -e 'ssh -p 2222' folder/ user@host:/path/

# Bandwidth limit
rsync -av --bwlimit=1000 folder/ user@host:/path/  # KB/s
```

Key rsync Options:

Option	Description
`-a`	Archive (preserves everything)
`-v`	Verbose
`-z`	Compress during transfer
`-P`	Progress + partial (resume)
`-n`	Dry run
`--delete`	Delete extra files at destination
`-e`	Specify remote shell

> **rsync is superior to scp** — it only transfers changed portions of files.

Command: `wget`

Download files from web

Syntax:

```
wget [OPTIONS] URL
```

Examples:

```
# Basic download
wget https://example.com/file.zip

# Save with different name
wget -O output.zip https://example.com/file. zip

# Download to directory
wget -P /tmp/ https://example.com/file. zip

# Continue interrupted download
wget -c https://example.com/large. iso

# Quiet mode
wget -q https://example.com/file.zip

# Download multiple files
wget -i urls.txt

# Recursive download (website mirror)
wget -r -l 2 https://example.com/
# -l = depth level

# Limit download rate
wget --limit-rate=500k https://example.com/file.zip

# Ignore SSL certificate errors
wget --no-check-certificate https://example.com/

# Add HTTP header
wget --header="Authorization: Bearer TOKEN" https://api.com/data

# POST request
wget --post-data="user=admin&pass=123" https://example.com/login
```

Command: `curl`

Transfer data from/to server (more versatile than wget)

Syntax:

```
curl [OPTIONS] URL
```

Examples:

```
# Basic GET request
curl https://example.com

# Save output
curl -o file.zip https://example.com/file. zip
curl -O https://example.com/file.zip    # Keep original name

# Follow redirects
curl -L https://example.com

# Show headers
curl -I https://example.com             # Headers only
curl -i https://example.com             # Headers + body

# Verbose output
curl -v https://example.com

# POST request
curl -X POST https://example.com/api
curl -X POST -d "param=value" https://example.com
curl -X POST -d @data.json https://example.com

# JSON data
curl -X POST -H "Content-Type: application/json" -d '{"key":"value"}' https://api.com

# Add headers
curl -H "Authorization: Bearer TOKEN" https://api.com
curl -H "Cookie: session=abc123" https://example.com

# Basic authentication
curl -u admin:password https://example.com

# Ignore SSL errors
curl -k https://self-signed.example.com

# Upload file
curl -F "file=@localfile.txt" https://example.com/upload

# Download with progress
curl -# -O https://example.com/large.file

# Set user agent
curl -A "Mozilla/5.0" https://example.com

# Use proxy
curl -x http://proxy:8080 https://example.com
curl --socks5 127.0.0.1:1080 https://example.com

# Rate limit
curl --limit-rate 100K https://example.com/file

# Timeout
curl --connect-timeout 5 -m 30 https://example.com
```

Key Terms

Term	Definition
Interface	Network connection point (eth0, wlan0, lo).
IP Address	Unique identifier on a network (IPv4: 192.168.1.1).

Term	Definition
Subnet Mask	Defines network/host portions of IP (/24 = 255.255.255.0).
Gateway	Router that forwards traffic to other networks.
DNS	Domain Name System — translates names to IPs.
Port	Logical endpoint for network services (22=SSH, 80=HTTP).
Socket	Combination of IP + Port + Protocol.
SSH	Secure Shell — encrypted remote access protocol.
ICMP	Internet Control Message Protocol (ping uses this).
ARP	Address Resolution Protocol — maps IP to MAC address.

❓ FAQs

Q: Should I use `ifconfig` or `ip`?

> Use `ip` — it's the modern standard and provides more features. Know `ifconfig` for older systems, but `ip` should be your default.

Q: Why can't I ping a host but can access it via HTTP?

> The host may be blocking ICMP (ping) packets via firewall while allowing TCP port 80/443. Many servers block ping for security. Try:
>
> ```
> curl -I http://host
> nc -zv host 80
> ```

Q: What's the difference between netstat and ss?

> `ss` is faster and provides more features — it reads directly from kernel. `netstat` is older and slower. Use `ss -tulpn` instead of `netstat -tulpn`.

Q: How do I check what's using a specific port?

> ```
> sudo lsof -i : 80
> sudo ss -tlpn | grep : 80
> sudo netstat -tlpn | grep :80
> ```

Q: What's the difference between `scp` and `rsync`?

> - `scp`: Simple file copy over SSH, copies everything every time
> - `rsync`: Syncs files, only transfers differences, supports resume
>
> Use `rsync` for large files, backups, and repeated transfers. Use `scp` for quick one-off copies.

🧪 Practice Tasks

Task 1: DNS Reconnaissance on Target Domain

Objective: Practice DNS enumeration techniques

Steps:

```
 # Choose a test target (legally!)
TARGET="scanme.nmap. org"

# Basic lookups
echo "=== A Records ==="
dig $TARGET A +short

echo "=== NS Records ==="
dig $TARGET NS +short

echo "=== MX Records ==="
dig $TARGET MX +short

echo "=== TXT Records ==="
dig $TARGET TXT +short

# Using nslookup
nslookup $TARGET

# Reverse lookup
IP=$(dig $TARGET +short | head -1)
dig -x $IP +short

# Trace DNS path
dig $TARGET +trace

# Using host
host -a $TARGET
```

Advanced Recon:

```
 # Attempt zone transfer (usually fails)
dig axfr @$(dig $TARGET NS +short | head -1) $TARGET

# Check for common subdomains
for sub in www mail ftp admin api dev staging; do
    result=$(dig +short $sub.$TARGET)
    if [ -n "$result" ]; then
        echo "$sub.$TARGET: $result"
    fi
done
```

Deliverable: Complete DNS report with all record types found

Task 2: Identify Open Ports and Active Connections

Objective: Practice network connection analysis

Steps:

```
# View all listening ports
sudo ss -tulpn

# Format output nicely
sudo ss -tulpn | column -t

# Active established connections
sudo ss -tpn state established

# View with lsof
sudo lsof -i -P -n | grep LISTEN

# Check specific ports
for port in 22 80 443 3306 5432; do
    if ss -tln | grep -q ":$port "; then
        echo "Port $port:  OPEN"
        sudo lsof -i : $port | tail -1
    else
        echo "Port $port: CLOSED"
    fi
done

# Monitor connections in real-time
watch -n 1 'ss -tpn state established'
```

Security Analysis:

```
# Find non-standard listening ports
sudo ss -tulpn | awk 'NR>1 && $5 ! ~ /: 22|:80|:443|:53/ {print}'

# Check for connections to suspicious IPs
sudo ss -tpn | grep -v "127.0.0.1\|192.168\|10.0"
```

Deliverable: List of all listening services with PID and process name

---

Task 3: Secure File Transfer with SSH Keys

Objective: Practice SSH key-based authentication and scp

Steps:

```
# Generate SSH key pair
ssh-keygen -t ed25519 -C "transfer-key" -f ~/.ssh/transfer_key
# Don't set passphrase for this exercise (or do for security)

# View keys
ls -la ~/.ssh/transfer_key*
cat ~/.ssh/transfer_key.pub

# Copy to remote (if you have another machine/VM)
# ssh-copy-id -i ~/.ssh/transfer_key.pub user@remote

# For local practice — setup local SSH
sudo systemctl start ssh
ssh-copy-id -i ~/. ssh/transfer_key.pub localhost

# Create test file
echo "Confidential data for transfer" > /tmp/evidence.txt

# Transfer file using key
scp -i ~/.ssh/transfer_key /tmp/evidence.txt localhost:/tmp/received.txt

# Verify
ssh -i ~/.ssh/transfer_key localhost "cat /tmp/received.txt"

# Transfer directory
mkdir -p /tmp/case_files
echo "File 1" > /tmp/case_files/file1.txt
echo "File 2" > /tmp/case_files/file2.txt

scp -i ~/.ssh/transfer_key -r /tmp/case_files localhost:/tmp/received_case/

# Using rsync
rsync -av -e "ssh -i ~/.ssh/transfer_key" /tmp/case_files/ localhost:/tmp/synced_case/
```

Deliverable: Screenshot showing successful key-based scp transfer

---

✅ Module 10 Checkpoint

Before moving to Module 11, confirm:

- ☐ Can view and configure network interfaces with `ip` command
- ☐ Can test connectivity with ping, traceroute, mtr
- ☐ Can perform DNS lookups with dig, nslookup, host
- ☐ Can view open ports and connections with ss, netstat, lsof
- ☐ Can connect via SSH with password and key authentication
- ☐ Can configure SSH keys and config file
- ☐ Can transfer files with scp, rsync, wget, curl
- ☐ Understand port forwarding and tunneling basics

Module 11: Shell Scripting Essentials

🎯 Introduction

Shell scripting transforms you from a Linux user into a **Linux automator**. Instead of typing commands repeatedly, you write scripts that do the work for you — reconnaissance automation, log analysis, backup routines, and custom tools.

In cybersecurity, scripting is non-negotiable. Custom scripts fill gaps between existing tools, automate tedious tasks, and enable rapid response during engagements. This module covers bash scripting from fundamentals to practical pentesting scripts.

11.1 Shebang and Script Basics

The Shebang (`#!`)

The **shebang** tells the system which interpreter to use:

```
#!/bin/bash          # Use bash
#!/bin/sh            # Use POSIX sh (more portable)
#!/usr/bin/env bash  # Find bash in PATH (most portable)
#!/usr/bin/env python3  # For Python scripts
```

Creating and Running Scripts

```
# Create script
nano myscript.sh

# Add shebang and code
#!/bin/bash
echo "Hello, World!"

# Save and exit (Ctrl+X, Y, Enter)

# Make executable
chmod +x myscript.sh

# Run script
./myscript.sh           # Current directory
bash myscript.sh        # Explicit interpreter
/full/path/myscript.sh  # Absolute path
```

Script Structure Template

```
#!/bin/bash
#
# Script Name:  example. sh
# Description:  Brief description of what this script does
# Author:  Loki
# Date: 2024-12-26
# Usage: ./example.sh [options] <arguments>
#

# Exit on error
set -e

# Variables
SCRIPT_DIR="$(cd "$(dirname "$0")" && pwd)"
LOG_FILE="/tmp/script. log"

# Functions
usage() {
    echo "Usage: $0 [-h] [-v] <target>"
    exit 1
}

# Main logic
main() {
    echo "Starting script..."
    # Your code here
}

# Run main
main "$@"
```

Useful Script Settings

```
#!/bin/bash
set -e          # Exit immediately on error
set -u          # Treat unset variables as error
set -o pipefail # Pipeline fails if any command fails
set -x          # Debug mode (print each command)

# Combined
set -euo pipefail
```

11.2 Variables & Data Types

Variable Declaration

```
# Basic assignment (NO spaces around =)
name="Loki"
age=20
path=/home/kali

# WRONG:
name = "Loki"    # Error: command not found

# Using variables
echo $name
echo "Hello, $name"
echo "Hello, ${name}"    # Braces for clarity
echo "Path is: ${path}/tools"

# Read-only variable
readonly PI=3.14159

# Unset variable
unset name
```

Variable Types

```
# Strings
greeting="Hello World"
filename='file.txt'      # Single quotes:  no expansion
message="User is $USER"  # Double quotes: expansion

# Numbers
count=10
port=8080

# Arrays
fruits=("apple" "banana" "cherry")
echo ${fruits[0]}        # apple
echo ${fruits[@]}        # All elements
echo ${#fruits[@]}       # Array length (3)

# Associative arrays (bash 4+)
declare -A user
user[name]="Loki"
user[role]="pentester"
echo ${user[name]}
```

Special Variables

Variable	Description
`$0`	Script name
`$1, $2, ...`	Positional arguments
`$#`	Number of arguments
`$@`	All arguments (as separate words)
`$*`	All arguments (as single string)
`$?`	Exit code of last command
`$$`	Current script's PID
`$!`	PID of last background process
`$_`	Last argument of previous command

```
#!/bin/bash
echo "Script name: $0"
echo "First argument:  $1"
echo "All arguments: $@"
echo "Number of arguments: $#"
```

String Operations

```
str="Hello World"

# Length
echo ${#str}              # 11

# Substring
echo ${str: 0:5}           # Hello (start: length)
echo ${str:6}             # World (from position 6)

# Replace
echo ${str/World/Loki}    # Hello Loki (first match)
echo ${str//l/L}          # HeLLo WorLd (all matches)

# Remove pattern
file="document.txt. bak"
echo ${file%. bak}         # document. txt (remove suffix)
echo ${file%.*}           # document. txt (remove last .*)
echo ${file%%.*}          # document (remove all . *)
echo ${file#*.}           # txt. bak (remove prefix)
echo ${file##*.}          # bak (remove longest prefix)

# Default values
echo ${undefined:-default}      # Use default if unset
echo ${undefined: =default}      # Set and use default if unset
echo ${defined: +replacement}    # Use replacement if set
```

Command Substitution

```
# Modern syntax (preferred)
current_date=$(date +%Y-%m-%d)
user_count=$(wc -l < /etc/passwd)
my_ip=$(hostname -I | awk '{print $1}')

# Legacy syntax (backticks)
current_date=`date +%Y-%m-%d`

# Nested substitution
files_in_dir=$(ls $(pwd) | wc -l)
```

Arithmetic

```
# Using $(( ))
result=$((5 + 3))
result=$((10 * 2))
result=$((100 / 5))
result=$((10 % 3))       # Modulo
result=$((2 ** 8))       # Exponent (256)

# Increment/decrement
count=0
((count++))
((count--))
((count += 5))

# Using let
let "x = 5 + 3"
let "x++"

# Using expr (legacy)
result=$(expr 5 + 3)

# Floating point (use bc)
result=$(echo "scale=2; 10 / 3" | bc)   # 3.33
```

11.3 User Input

Command: `read`

Read user input into variables

Syntax:

```
read [OPTIONS] VARIABLE(S)
```

Examples:

```
# Basic input
echo "Enter your name:"
read name
echo "Hello, $name!"

# Prompt on same line
read -p "Enter your name: " name

# Silent input (passwords)
read -sp "Enter password: " password
echo    # New line after silent input

# Timeout
read -t 5 -p "Quick!  Enter value: " value

# Default value
read -p "Port [8080]: " port
port=${port:-8080}

# Read into array
read -a items -p "Enter items (space-separated): "
echo "First item: ${items[0]}"

# Read specific number of characters
read -n 1 -p "Continue? (y/n): " answer
echo

# Read from file
while read line; do
    echo "Processing: $line"
done < file. txt
```

Common Options:

Option	Description
`-p "prompt"`	Display prompt
`-s`	Silent mode (hide input)
`-t SECONDS`	Timeout
`-n NUM`	Read NUM characters
`-a ARRAY`	Read into array
`-r`	Raw mode (don't interpret backslashes)

Validating Input

```
#!/bin/bash
# Input validation example

while true; do
    read -p "Enter a number (1-100): " num

    # Check if number
    if !  [[ "$num" =~ ^[0-9]+$ ]]; then
        echo "Error: Not a number"
        continue
    fi

    # Check range
    if (( num < 1 || num > 100 )); then
        echo "Error: Out of range"
        continue
    fi

    break
done

echo "Valid input:  $num"
```

11.4 Conditionals

`if` Statement

Syntax:

```
if [ condition ]; then
    # commands
elif [ condition ]; then
    # commands
else
    # commands
fi
```

Examples:

```
#!/bin/bash

# Basic if
if [ "$1" = "hello" ]; then
    echo "Hello back!"
fi

# if-else
if [ -f "/etc/passwd" ]; then
    echo "File exists"
else
    echo "File not found"
fi

# if-elif-else
score=85

if [ $score -ge 90 ]; then
    echo "Grade:  A"
elif [ $score -ge 80 ]; then
    echo "Grade: B"
elif [ $score -ge 70 ]; then
    echo "Grade: C"
else
    echo "Grade: F"
fi
```

Test Operators

File Tests:

Operator	Description
`-e FILE`	Exists
`-f FILE`	Regular file
`-d FILE`	Directory
`-r FILE`	Readable
`-w FILE`	Writable
`-x FILE`	Executable
`-s FILE`	Size > 0
`-L FILE`	Symbolic link

String Tests:

Operator	Description
`-z STRING`	Empty string
`-n STRING`	Non-empty string
`STRING1 = STRING2`	Equal
`STRING1 != STRING2`	Not equal

Numeric Tests:

Operator	Description

Operator	Description
`-eq`	Equal
`-ne`	Not equal
`-lt`	Less than
`-le`	Less than or equal
`-gt`	Greater than
`-ge`	Greater than or equal

Logical Operators:

Operator	Description
`!`	NOT
`-a` or `&&`	AND
`-o` or `||`	OR

`[ ]` vs `[[ ]]`

```
# [ ] - POSIX compatible, limited
if [ "$var" = "value" ]; then echo "match"; fi

# [[ ]] - Bash extended, more features
if [[ "$var" == "value" ]]; then echo "match"; fi

# [[ ]] supports:
# - Pattern matching
if [[ "$file" == *.txt ]]; then echo "Text file"; fi

# - Regex matching
if [[ "$email" =~ ^[a-z]+@[a-z]+\.[a-z]+$ ]]; then echo "Valid email"; fi

# - No word splitting (safer without quotes)
if [[ $var == value ]]; then echo "match"; fi
```

`case` Statement

Syntax:

```
case $variable in
    pattern1)
        commands
        ;;
    pattern2|pattern3)
        commands
        ;;
    *)
        default commands
        ;;
esac
```

Example:

```
#!/bin/bash

read -p "Enter command (start/stop/restart): " cmd

case $cmd in
    start)
        echo "Starting service..."
        ;;
    stop)
        echo "Stopping service..."
        ;;
    restart|reload)
        echo "Restarting service..."
        ;;
    *)
        echo "Unknown command:  $cmd"
        echo "Usage: start|stop|restart"
        exit 1
        ;;
esac
```

Pattern Matching in case:

```
case $file in
    *. txt)     echo "Text file" ;;
    *.sh)      echo "Shell script" ;;
    *.tar.gz)  echo "Compressed archive" ;;
    [0-9]*)    echo "Starts with number" ;;
    *)         echo "Unknown type" ;;
esac
```

11.5 Loops

`for` Loop

Syntax:

```
for variable in list; do
    commands
done
```

Examples:

```
# Loop over values
for fruit in apple banana cherry; do
    echo "Fruit: $fruit"
done

# Loop over range
for i in {1..10}; do
    echo "Number: $i"
done

# Range with step
for i in {0..100..10}; do
    echo "Count: $i"
done

# Loop over files
for file in *.txt; do
    echo "Processing: $file"
done

# Loop over command output
for user in $(cat /etc/passwd | cut -d: -f1); do
    echo "User:  $user"
done

# C-style for loop
for ((i=0; i<10; i++)); do
    echo "Index: $i"
done

# Loop over array
servers=("web1" "web2" "db1")
for server in "${servers[@]}"; do
    echo "Checking $server..."
done

# Loop with index
arr=("a" "b" "c")
for i in "${! arr[@]}"; do
    echo "Index $i: ${arr[$i]}"
done
```

`while` Loop

Syntax:

```
while [ condition ]; do
    commands
done
```

Examples:

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

# Read file line by line
while IFS= read -r line; do
    echo "Line:  $line"
done < file.txt

# Infinite loop (until break)
while true; do
    read -p "Enter 'quit' to exit: " input
    if [ "$input" = "quit" ]; then
        break
    fi
    echo "You entered:  $input"
done

# Process while condition true
while pgrep -x "process_name" > /dev/null; do
    echo "Process still running..."
    sleep 5
done
echo "Process stopped!"

# Read from command output
find /var/log -name "*.log" | while read -r logfile; do
    echo "Log: $logfile"
done
```

`until` Loop

Syntax:

```
until [ condition ]; do
    commands
done
```

Example:

```
# Wait until service is up
until nc -z localhost 80 2>/dev/null; do
    echo "Waiting for web server..."
    sleep 2
done
echo "Web server is up!"
```

Loop Control

```
# break - exit loop
for i in {1.. 10}; do
    if [ $i -eq 5 ]; then
        break
    fi
    echo $i
done
# Output: 1 2 3 4

# continue - skip to next iteration
for i in {1..5}; do
    if [ $i -eq 3 ]; then
        continue
    fi
    echo $i
done
# Output: 1 2 4 5

# break n - break out of n nested loops
for i in {1..3}; do
    for j in {1.. 3}; do
        if [ $j -eq 2 ]; then
            break 2   # Break out of both loops
        fi
        echo "$i,$j"
    done
done
```

11.6 Functions

Function Definition

```
# Method 1 (preferred)
function_name() {
    commands
}

# Method 2
function function_name {
    commands
}
```

Examples:

```bash
#!/bin/bash

# Simple function
greet() {
    echo "Hello, World!"
}

# Call function
greet

# Function with parameters
greet_user() {
    local name=$1      # Local variable
    local greeting=${2:-"Hello"}   # Default value
    echo "$greeting, $name!"
}

greet_user "Loki"
greet_user "Loki" "Welcome"

# Function with return value
is_valid_ip() {
    local ip=$1
    if [[ $ip =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then
        return 0   # Success (true)
    else
        return 1   # Failure (false)
    fi
}

if is_valid_ip "192.168.1.1"; then
    echo "Valid IP"
else
    echo "Invalid IP"
fi

# Function returning string (via echo)
get_timestamp() {
    echo $(date +%Y%m%d_%H%M%S)
}

timestamp=$(get_timestamp)
echo "Timestamp:  $timestamp"
```

Local Variables

```bash
#!/bin/bash

global_var="I'm global"

my_function() {
    local local_var="I'm local"
    global_var="Modified global"
    echo "Inside:  $local_var, $global_var"
}

my_function
echo "Outside: $global_var"
# local_var is not accessible here
```

Function Library

Create reusable functions in a library file:

```
# lib/utils.sh
#!/bin/bash

log_info() {
    echo "[INFO] $(date '+%Y-%m-%d %H:%M:%S') - $1"
}

log_error() {
    echo "[ERROR] $(date '+%Y-%m-%d %H:%M:%S') - $1" >&2
}

die() {
    log_error "$1"
    exit ${2:-1}
}

require_root() {
    if [ "$EUID" -ne 0 ]; then
        die "This script must be run as root"
    fi
}
```

```
# main_script.sh
#!/bin/bash
source ./lib/utils.sh

require_root
log_info "Script started"
# ...
```

11.7 Exit Codes & Error Handling

Exit Codes

Code	Meaning
`0`	Success
`1`	General error
`2`	Misuse of command
`126`	Permission denied
`127`	Command not found
`128+N`	Killed by signal N
`130`	Ctrl+C (SIGINT)

```
#!/bin/bash

# Set exit code
if [ !  -f "$1" ]; then
    echo "File not found: $1"
    exit 1
fi

# Check last command's exit code
grep "pattern" file.txt
if [ $? -eq 0 ]; then
    echo "Pattern found"
else
    echo "Pattern not found"
fi

# Inline check
grep "pattern" file.txt && echo "Found" || echo "Not found"
```

Error Handling

```
#!/bin/bash
set -e          # Exit on any error
set -o pipefail # Pipeline fails on any error

# Trap errors
trap 'echo "Error on line $LINENO"; exit 1' ERR

# Trap exit (cleanup)
cleanup() {
    echo "Cleaning up..."
    rm -f /tmp/tempfile.*
}
trap cleanup EXIT

# Trap signals
trap 'echo "Interrupted! "; exit 130' INT TERM

# Safe command execution with error message
run_cmd() {
    "$@" || { echo "Command failed:  $*"; exit 1; }
}

run_cmd ls /nonexistent
```

Logging and Debugging

```
#!/bin/bash

# Logging to file and stdout
LOG_FILE="/tmp/script.log"

log() {
    local level=$1
    shift
    local message="$*"
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')
    echo "[$timestamp] [$level] $message" | tee -a "$LOG_FILE"
}

log INFO "Script started"
log WARN "This is a warning"
log ERROR "This is an error"

# Debug mode
DEBUG=${DEBUG:-false}
debug() {
    if [ "$DEBUG" = true ]; then
        echo "[DEBUG] $*" >&2
    fi
}

DEBUG=true
debug "Variable x = $x"
```

11.8 Practical Scripts

Script 1: Subdomain Enumeration

```
#!/bin/bash
#
# subdomain_enum.sh - Automated subdomain enumeration
# Usage: ./subdomain_enum.sh <domain>
#

set -euo pipefail

# Colors
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

# Check arguments
if [ $# -ne 1 ]; then
    echo "Usage: $0 <domain>"
    echo "Example: $0 example.com"
    exit 1
fi

DOMAIN=$1
OUTPUT_DIR="./recon/${DOMAIN}"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
RESULTS_FILE="${OUTPUT_DIR}/subdomains_${TIMESTAMP}.txt"

# Create output directory
mkdir -p "$OUTPUT_DIR"

echo -e "${GREEN}[*] Starting subdomain enumeration for: ${DOMAIN}${NC}"
```

```
# Method 1: Common subdomains wordlist
echo -e "${YELLOW}[*] Checking common subdomains...${NC}"
COMMON_SUBS="www mail ftp admin api dev staging test blog shop app portal"

for sub in $COMMON_SUBS; do
    result=$(dig +short "${sub}.${DOMAIN}" 2>/dev/null)
    if [ -n "$result" ]; then
        echo -e "${GREEN}[+] Found: ${sub}.${DOMAIN} -> ${result}${NC}"
        echo "${sub}.${DOMAIN}" >> "$RESULTS_FILE"
    fi
done

# Method 2: DNS queries
echo -e "${YELLOW}[*] Querying DNS records...${NC}"

# Get nameservers
echo -e "  [*] NS Records:"
dig +short NS "$DOMAIN" | while read ns; do
    echo "      $ns"
done

# Get MX records (often reveal subdomains)
echo -e "  [*] MX Records:"
dig +short MX "$DOMAIN" | while read mx; do
    echo "      $mx"
done

# Method 3: Certificate Transparency (if curl available)
if command -v curl &> /dev/null; then
    echo -e "${YELLOW}[*] Checking Certificate Transparency logs...${NC}"
    curl -s "https://crt.sh/? q=%. ${DOMAIN}&output=json" 2>/dev/null | \
        grep -oP '"name_value": "\K[^"]+' | \
        sort -u | \
        while read subdomain; do
            echo -e "${GREEN}[+] CT Found: ${subdomain}${NC}"
            echo "$subdomain" >> "$RESULTS_FILE"
        done
fi

# Deduplicate results
if [ -f "$RESULTS_FILE" ]; then
    sort -u "$RESULTS_FILE" -o "$RESULTS_FILE"
    echo ""
    echo -e "${GREEN}[*] Results saved to: ${RESULTS_FILE}${NC}"
    echo -e "${GREEN}[*] Total unique subdomains: $(wc -l < "$RESULTS_FILE")${NC}"
else
    echo -e "${RED}[! ] No subdomains found${NC}"
fi
```

Script 2: Backup Script

```
#!/bin/bash
#
# backup. sh - Automated backup with compression and rotation
# Usage: ./backup.sh <source_dir> [backup_dir]
#

set -euo pipefail

# Configuration
SOURCE_DIR="${1:? Usage: $0 <source_dir> [backup_dir]}"
BACKUP_DIR="${2:-/backup}"
MAX_BACKUPS=7  # Keep last 7 backups
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
HOSTNAME=$(hostname)

# Validate source
if [ ! -d "$SOURCE_DIR" ]; then
    echo "Error: Source directory does not exist: $SOURCE_DIR"
    exit 1
fi

# Create backup directory
mkdir -p "$BACKUP_DIR"

# Backup filename
SOURCE_NAME=$(basename "$SOURCE_DIR")
BACKUP_FILE="${BACKUP_DIR}/${HOSTNAME}_${SOURCE_NAME}_${TIMESTAMP}.tar.gz"

# Create backup
echo "[*] Creating backup of: $SOURCE_DIR"
echo "[*] Backup file: $BACKUP_FILE"

tar -czf "$BACKUP_FILE" -C "$(dirname "$SOURCE_DIR")" "$(basename "$SOURCE_DIR")"

# Verify backup
if [ -f "$BACKUP_FILE" ]; then
    SIZE=$(du -h "$BACKUP_FILE" | cut -f1)
    echo "[+] Backup created successfully: $SIZE"

    # Create checksum
    sha256sum "$BACKUP_FILE" > "${BACKUP_FILE}. sha256"
    echo "[+] Checksum saved:  ${BACKUP_FILE}.sha256"
else
    echo "[-] Backup failed!"
    exit 1
fi

# Rotate old backups
echo "[*] Rotating old backups (keeping last $MAX_BACKUPS)..."
ls -t "${BACKUP_DIR}/${HOSTNAME}_${SOURCE_NAME}_"*. tar.gz 2>/dev/null | \
    tail -n +$((MAX_BACKUPS + 1)) | \
    while read old_backup; do
        echo "    Removing: $(basename "$old_backup")"
        rm -f "$old_backup" "${old_backup}.sha256"
    done

echo "[*] Backup complete!"

# Show current backups
echo ""
echo "Current backups:"
ls -lh "${BACKUP_DIR}/${HOSTNAME}_${SOURCE_NAME}_"*.tar. gz 2>/dev/null || echo "  None"
```

Script 3: Port Scanner

```
#!/bin/bash
#
# portscan.sh - Simple bash port scanner
# Usage: ./portscan.sh <target> [port_range]
#

set -uo pipefail

# Colors
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
CYAN='\033[0;36m'
NC='\033[0m'

# Check arguments
TARGET="${1:?Usage: $0 <target> [start_port-end_port]}"
PORT_RANGE="${2:-1-1000}"

# Parse port range
START_PORT=$(echo "$PORT_RANGE" | cut -d'-' -f1)
END_PORT=$(echo "$PORT_RANGE" | cut -d'-' -f2)

# Validate
if !  [[ "$START_PORT" =~ ^[0-9]+$ ]] || !  [[ "$END_PORT" =~ ^[0-9]+$ ]]; then
    echo "Error:  Invalid port range"
    exit 1
fi

echo -e "${CYAN}====================================${NC}"
echo -e "${CYAN}   Bash Port Scanner v1.0${NC}"
echo -e "${CYAN}====================================${NC}"
echo -e "${YELLOW}Target: ${NC} $TARGET"
echo -e "${YELLOW}Ports: ${NC}  $START_PORT - $END_PORT"
echo -e "${CYAN}====================================${NC}"
echo ""

# Check if target is reachable
if !  ping -c 1 -W 2 "$TARGET" &>/dev/null; then
    echo -e "${YELLOW}[! ] Host might be down or blocking ICMP${NC}"
fi

# Scan ports
OPEN_PORTS=()

echo -e "${YELLOW}[*] Scanning ports... ${NC}"

for ((port=START_PORT; port<=END_PORT; port++)); do
    # Progress indicator (every 100 ports)
    if (( port % 100 == 0 )); then
        echo -ne "\r[*] Progress: $port / $END_PORT   "
    fi

    # TCP connect scan using /dev/tcp (bash built-in)
    if timeout 1 bash -c "echo >/dev/tcp/$TARGET/$port" 2>/dev/null; then
        # Get service name if possible
        service=$(grep -w "$port/tcp" /etc/services 2>/dev/null | head -1 | awk '{print $1}')
        service=${service:-"unknown"}

        echo -e "\r${GREEN}[+] Port $port OPEN ($service)${NC}          "
        OPEN_PORTS+=("$port")
    fi
done

# Summary
```

```
echo ""
echo -e "${CYAN}====================================${NC}"
echo -e "${CYAN}   Scan Complete${NC}"
echo -e "${CYAN}====================================${NC}"

if [ ${#OPEN_PORTS[@]} -gt 0 ]; then
    echo -e "${GREEN}Open ports found: ${#OPEN_PORTS[@]}${NC}"
    echo -e "${GREEN}Ports: ${OPEN_PORTS[*]}${NC}"
else
    echo -e "${RED}No open ports found in range${NC}"
fi
```

Key Terms

Term	Definition
Shebang	`#!` line specifying the script interpreter.
Variable	Named storage for data in scripts.
Exit Code	Numeric value returned by command (0 = success).
stdin/stdout/stderr	Standard input (0), output (1), error (2) streams.
Pipe	`|` connects stdout of one command to stdin of another.
Subshell	Child shell created by `$()` or `()`.
Function	Reusable block of code within a script.
Local Variable	Variable scoped to a function only.

FAQs

Q: Why does my script work when run with `bash script.sh` but not `./script.sh`?

> Either missing shebang or not executable:
>
> ```
> chmod +x script.sh
> # Ensure first line is:  #!/bin/bash
> ```

**Q: How do I pass arguments to a script? **

> Use positional parameters:
>
> ```
> ./script.sh arg1 arg2
> # In script:  $1 = arg1, $2 = arg2, $@ = all args
> ```

**Q: Why do I need quotes around variables? **

> Without quotes, word splitting occurs:
>
> ```
> file="my file.txt"
> cat $file    # Error:  tries cat "my" and "file.txt"
> cat "$file"  # Correct:  cat "my file. txt"
> ```

Q: What's the difference between `$@` and `$*`?

> - `"$@"` expands to separate quoted arguments: `"arg1" "arg2" "arg3"`
> - `"$*"` expands to single string: `"arg1 arg2 arg3"`
>
> Use `"$@"` to preserve argument boundaries in loops.

Q: How do I debug a bash script?

```
bash -x script.sh       # Print each command
set -x                  # Enable debug in script
set +x                  # Disable debug
```

Practice Tasks

Task 1: Subdomain Enumeration Script

Objective: Automate subdomain discovery

Use the script from Section 11.8 or create your own:

```
# Test with safe domain
chmod +x subdomain_enum.sh
./subdomain_enum.sh scanme.nmap.org
```

Deliverable: Screenshot of script running with discovered subdomains

---

Task 2: Backup Script with Timestamps

Objective: Create automated backup routine

Steps:

```
# Create test data
mkdir -p ~/test_data
echo "Important file 1" > ~/test_data/file1.txt
echo "Important file 2" > ~/test_data/file2.txt

# Run backup script
chmod +x backup.sh
./backup.sh ~/test_data /tmp/backups

# Verify backup
ls -la /tmp/backups/
tar -tzf /tmp/backups/*. tar.gz
```

Deliverable: Screenshot showing backup creation and contents listing

---

Task 3: Bash Port Scanner

Objective: Build functional port scanner

Steps:

```
chmod +x portscan.sh

# Scan localhost
./portscan.sh 127.0.0.1 1-100

# Scan safe target
./portscan.sh scanme. nmap.org 20-100
```

Deliverable: Screenshot of port scan results

Module 11 Checkpoint

Before moving to Module 12, confirm:

- ☐ Can create executable scripts with proper shebang
- ☐ Understand variables, arrays, and special parameters
- ☐ Can read user input and validate it
- ☐ Can use conditionals (if/else, case)
- ☐ Can write loops (for, while, until)
- ☐ Can create and use functions
- ☐ Understand exit codes and error handling
- ☐ Can write practical automation scripts

Module 12: System Monitoring & Logs

Introduction

System monitoring and log analysis are essential for **system administration, security monitoring, and incident response**. Understanding system health, resource usage, and log files allows you to detect anomalies, troubleshoot issues, and identify security incidents.

This module covers system information commands, resource monitoring, log file analysis, and task scheduling — the foundation for maintaining healthy and secure systems.

12.1 System Information

Command: `uname`

Print system information

```
 # All information
uname -a
# Linux kali 6.1.0-kali5-amd64 #1 SMP PREEMPT_DYNAMIC x86_64 GNU/Linux

# Kernel name
uname -s    # Linux

# Kernel release
uname -r    # 6.1.0-kali5-amd64

# Kernel version
uname -v    # #1 SMP PREEMPT_DYNAMIC

# Machine hardware
uname -m    # x86_64

# Processor type
uname -p    # x86_64

# Operating system
uname -o    # GNU/Linux
```

Command: `hostname`

Show or set system hostname

```
# Display hostname
hostname

# Short hostname
hostname -s

# FQDN (fully qualified domain name)
hostname -f

# IP address
hostname -I

# All IP addresses
hostname -i

# Set hostname (temporary)
sudo hostname newhostname

# Set hostname (permanent)
sudo hostnamectl set-hostname newhostname
```

Command: `hostnamectl`

Control system hostname and metadata

```
hostnamectl
#    Static hostname: kali
#          Icon name: computer-vm
#            Chassis: vm
#         Machine ID: abc123...
#            Boot ID: def456...
#   Operating System: Kali GNU/Linux Rolling
#             Kernel: Linux 6.1.0-kali5-amd64
#       Architecture: x86-64
```

Command: `uptime`

Show how long system has been running

```
uptime
# 10:30:00 up 5 days, 12:45,  2 users,  load average: 0.15, 0.10, 0.05

# Uptime only
uptime -p
# up 5 days, 12 hours, 45 minutes

# Since boot
uptime -s
# 2024-12-20 21:45:00
```

Load Average Explained:

```
load average: 0.15, 0.10, 0.05
              |     |     |
              |     |     └── 15-minute average
              |     └── 5-minute average
              └── 1-minute average

# Load = number of processes waiting for CPU
# For 4-core system: load < 4. 0 is healthy
```

Command: `lsb_release`

Distribution information

```
 lsb_release -a
# Distributor ID:  Kali
# Description:     Kali GNU/Linux Rolling
# Release:        2024. 4
# Codename:       kali-rolling

# Short version
lsb_release -rs    # 2024.4
```

Other System Info Commands

```
 # Detailed OS info
cat /etc/os-release

# Kernel parameters
cat /proc/version
cat /proc/cmdline

# CPU info
lscpu
cat /proc/cpuinfo

# Memory info
cat /proc/meminfo

# Hardware info
sudo lshw -short
sudo dmidecode -t system

# PCI devices
lspci

# USB devices
lsusb

# Block devices
lsblk

# System date/time
date
timedatectl
```

12.2 Resource Monitoring

Command: `free`

Display memory usage

```
 # Human-readable
free -h
#                total        used        free      shared  buff/cache   available
# Mem:            15Gi       3.2Gi        10Gi       256Mi       2.0Gi        11Gi
# Swap:          2. 0Gi          0B       2.0Gi

# Continuous monitoring
free -h -s 2    # Update every 2 seconds

# Show totals
free -h -t

# Wide format
free -hw
```

Memory Fields:

Field	Description
`total`	Total physical memory
`used`	Memory in use by processes
`free`	Completely unused memory
`shared`	Memory used by tmpfs
`buff/cache`	Buffer + cache memory
`available`	Memory available for new processes

> **Focus on `available`, not `free`**. Linux uses free RAM for cache, which is released when needed.

Command: `vmstat`

Virtual memory statistics

```
 # One-time snapshot
vmstat

# Continuous monitoring (every 2 seconds)
vmstat 2

# With timestamp
vmstat -t 2

# Disk statistics
vmstat -d

# Active/inactive memory
vmstat -a

# Output:
# procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
#  r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
#  1  0      0 10240000 128000 2048000  0    0     5    10  100  200  2  1 97  0  0
```

vmstat Fields:

Field	Description
`r`	Runnable processes (waiting for CPU)

Field	Description
`b`	Blocked processes (waiting for I/O)
`swpd`	Virtual memory used
`si/so`	Swap in/out (should be 0)
`bi/bo`	Blocks read/written per second
`us`	User CPU time %
`sy`	System CPU time %
`id`	Idle CPU %
`wa`	I/O wait %

Command: `iostat`

I/O and CPU statistics

```
 # Install
sudo apt install sysstat

# Basic output
iostat

# Extended stats with human-readable
iostat -xh

# Specific device
iostat -xh /dev/sda

# Continuous monitoring
iostat -xh 2      # Every 2 seconds

# CPU stats only
iostat -c

# Disk stats only
iostat -d
```

Command: `sar`

System Activity Reporter (historical data)

```
# Current CPU usage
sar -u 1 5     # 1-second intervals, 5 times

# Memory usage
sar -r 1 5

# Disk I/O
sar -b 1 5

# Network stats
sar -n DEV 1 5

# Historical data (if sysstat enabled)
sar -u -f /var/log/sysstat/sa$(date +%d)

# Enable sysstat collection
sudo systemctl enable sysstat
sudo systemctl start sysstat
```

Quick Monitoring Commands

```
# CPU usage (quick)
grep "cpu " /proc/stat | awk '{print "CPU: " ($2+$4)*100/($2+$4+$5) "%"}'

# Top processes by CPU
ps aux --sort=-%cpu | head -10

# Top processes by memory
ps aux --sort=-%mem | head -10

# Disk I/O in real-time
sudo iotop

# Network I/O in real-time
sudo iftop
sudo nethogs

# Process tree
pstree -p

# Process monitor (enhanced)
htop
btop
```

12.3 Log Files

Log File Locations

Log File	Description
`/var/log/syslog`	General system log (Debian/Ubuntu)
`/var/log/messages`	General system log (RHEL/CentOS)
`/var/log/auth.log`	Authentication logs (logins, sudo)
`/var/log/secure`	Auth logs (RHEL/CentOS)

Log File	Description
`/var/log/kern.log`	Kernel messages
`/var/log/dmesg`	Boot messages
`/var/log/boot.log`	Boot process log
`/var/log/cron`	Cron job logs
`/var/log/apt/`	APT package manager logs
`/var/log/dpkg.log`	Package installation log
`/var/log/apache2/`	Apache web server logs
`/var/log/nginx/`	Nginx web server logs
`/var/log/mysql/`	MySQL database logs
`/var/log/fail2ban. log`	Fail2ban logs
`/var/log/ufw.log`	UFW firewall logs

Log Viewing Commands

```
# View entire log
cat /var/log/syslog

# Last entries
tail /var/log/syslog
tail -n 50 /var/log/syslog

# Real-time monitoring
tail -f /var/log/syslog
tail -F /var/log/auth.log    # Follows rotations

# Multiple files
tail -f /var/log/syslog /var/log/auth. log

# Filter while watching
tail -f /var/log/auth.log | grep "Failed"

# Page through log
less /var/log/syslog
# Use / to search, n for next, q to quit

# Search in logs
grep "error" /var/log/syslog
grep -i "failed" /var/log/auth.log
grep -E "error|warning|critical" /var/log/syslog

# With context
grep -B 2 -A 2 "error" /var/log/syslog

# By date
grep "Dec 25" /var/log/auth.log
```

Command: `journalctl`

Query systemd journal

```
# All logs
journalctl

# Follow (real-time)
journalctl -f

# Since boot
journalctl -b
journalctl -b -1     # Previous boot

# By unit/service
journalctl -u ssh
journalctl -u apache2
journalctl -u nginx --since "1 hour ago"

# By priority
journalctl -p err       # Errors only
journalctl -p warning   # Warnings and above

# Time range
journalctl --since "2024-12-25 10:00"
journalctl --since "1 hour ago"
journalctl --since "yesterday"
journalctl --since "2024-12-25" --until "2024-12-26"

# Kernel messages
journalctl -k

# By executable
journalctl /usr/bin/sudo

# Output formats
journalctl -o json-pretty
journalctl -o short-precise

# Disk usage
journalctl --disk-usage

# Clean old logs
sudo journalctl --vacuum-time=7d
sudo journalctl --vacuum-size=500M

# Show specific number of lines
journalctl -n 50
```

Priority Levels

Priority	Keyword	Description
0	`emerg`	System unusable
1	`alert`	Immediate action needed
2	`crit`	Critical conditions
3	`err`	Error conditions
4	`warning`	Warning conditions
5	`notice`	Normal but significant
6	`info`	Informational
7	`debug`	Debug messages

Priority	Keyword	Description

Security Log Analysis

```
# Failed login attempts
grep "Failed password" /var/log/auth.log
journalctl _COMM=sshd | grep "Failed"

# Successful logins
grep "Accepted" /var/log/auth.log

# Sudo usage
grep "sudo" /var/log/auth.log
grep "COMMAND=" /var/log/auth.log

# User creation/modification
grep -E "useradd|usermod|userdel" /var/log/auth. log

# SSH connections
grep "sshd" /var/log/auth. log

# Count failed attempts by IP
grep "Failed password" /var/log/auth.log | \
    grep -oE '[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+' | \
    sort | uniq -c | sort -rn | head -10

# Brute force detection
journalctl -u ssh --since "1 hour ago" | grep "Failed" | wc -l
```

12.4 Cron Jobs & Scheduling

Understanding Cron

Cron runs scheduled tasks automatically.

Cron Syntax

```
# ┌──────────────── minute (0-59)
# │ ┌──────────────── hour (0-23)
# │ │ ┌──────────────── day of month (1-31)
# │ │ │ ┌──────────────── month (1-12)
# │ │ │ │ ┌──────────────── day of week (0-6, Sunday=0)
# │ │ │ │ │
# * * * * * command to execute
```

Common Cron Patterns

Pattern	Description
`* * * * *`	Every minute
`0 * * * *`	Every hour
`0 0 * * *`	Daily at midnight
`0 0 * * 0`	Weekly (Sunday midnight)
`0 0 1 * *`	Monthly (1st at midnight)
`*/5 * * * *`	Every 5 minutes

Pattern	Description
`0 */2 * * *`	Every 2 hours
`0 9-17 * * 1-5`	Hourly 9-5, weekdays
`0 0 * * 1-5`	Weekdays at midnight

Managing Crontab

```
# Edit current user's crontab
crontab -e

# List current user's crontab
crontab -l

# Remove all cron jobs
crontab -r

# Edit another user's crontab (as root)
sudo crontab -e -u username

# View another user's crontab
sudo crontab -l -u username
```

Example Crontabs

```
# Edit crontab
crontab -e

# Add these lines:

# Daily backup at 2 AM
0 2 * * * /home/kali/scripts/backup.sh >> /var/log/backup.log 2>&1

# Every 15 minutes - check disk space
*/15 * * * * /home/kali/scripts/check_disk. sh

# Weekly security scan on Sunday at 3 AM
0 3 * * 0 /home/kali/scripts/weekly_scan.sh

# Every hour - sync logs
0 * * * * rsync -av /var/log/ /backup/logs/ 2>/dev/null

# Monthly report on 1st at 8 AM
0 8 1 * * /home/kali/scripts/monthly_report. sh | mail -s "Monthly Report" admin@example.com
```

Special Cron Strings

```
@reboot    # Run once at startup
@yearly    # 0 0 1 1 * (annually)
@monthly   # 0 0 1 * *
@weekly    # 0 0 * * 0
@daily     # 0 0 * * * (midnight)
@hourly    # 0 * * * *
```

System Cron Directories

```
# System-wide crontabs
/etc/crontab

# Cron directories (scripts here run automatically)
/etc/cron.hourly/
/etc/cron.daily/
/etc/cron.weekly/
/etc/cron. monthly/

# Per-user crontabs
/var/spool/cron/crontabs/
```

Cron Output and Debugging

```
# Redirect output to log
* * * * * /script. sh >> /var/log/script.log 2>&1

# Discard output
* * * * * /script.sh > /dev/null 2>&1

# Email output (if mail configured)
MAILTO="admin@example.com"
0 * * * * /script.sh

# Debug:  Check cron logs
grep CRON /var/log/syslog
journalctl -u cron
```

Command: `at`

Schedule one-time tasks

```
# Install
sudo apt install at
sudo systemctl start atd

# Schedule command
at 10:30
> /home/kali/scripts/task.sh
> Ctrl+D

# Schedule for specific date
at 10:30 12/25/2024

# Schedule relative time
at now + 5 minutes
at now + 2 hours
at now + 1 day

# From command line
echo "/home/kali/scripts/task.sh" | at now + 10 minutes

# List pending jobs
atq

# Remove job
atrm JOB_NUMBER

# View job details
at -c JOB_NUMBER
```

Key Terms

Term	Definition
Load Average	Number of processes waiting for CPU, averaged over 1/5/15 minutes.
Swap	Disk space used as virtual memory when RAM is full.
Buffer/Cache	Memory used to speed up disk operations, released when needed.
Journal	Systemd's binary log storage, queried with journalctl.
Cron	Time-based job scheduler running tasks automatically.
Daemon	Background service running continuously.
Syslog	Traditional logging system and protocol.
Log Rotation	Automatic archiving and cleanup of old logs.

FAQs

Q: Why does `free` show very little "free" memory?

> Linux uses free RAM for disk caching, which improves performance. This memory is released when applications need it. Look at the `available` column, not `free`, for usable memory.

Q: What's the difference between /var/log/syslog and journalctl?

> - `syslog`: Traditional text log file, managed by rsyslog/syslog-ng
> - `journalctl`: Systemd journal, binary format with better querying
>
> Modern systems use both. Some services log only to journal.

Q: How do I make a cron job run with environment variables?

> Cron runs with minimal environment. Options:
>
> ```
> # Define in crontab
> PATH=/usr/local/bin:/usr/bin:/bin
> HOME=/home/kali
> 0 * * * * /script.sh
>
> # Or source profile in script
> #!/bin/bash
> source /home/kali/. bashrc
> # rest of script
> ```

Q: Why isn't my cron job running?

> Common issues:
>
> 1. Script not executable: `chmod +x script.sh`
> 2. Wrong path: Use absolute paths
> 3. Environment: Cron has minimal env
> 4. Output hidden: Add `>> /tmp/cron. log 2>&1`
> 5. Check logs: `grep CRON /var/log/syslog`

Q: How do I monitor a specific process's resource usage?

```
# Using top
top -p $(pgrep process_name)

# Using ps
ps -p PID -o pid,%cpu,%mem,cmd

# Using watch
watch -n 1 'ps -p PID -o %cpu,%mem'
```

Practice Tasks

Task 1: Monitor Failed Login Attempts in Real-Time

Objective: Detect potential brute force attacks

Steps:

```
# Method 1: tail with grep
sudo tail -f /var/log/auth.log | grep --line-buffered "Failed"

# Method 2: journalctl
sudo journalctl -u ssh -f | grep --line-buffered "Failed"

# Method 3: Custom monitoring script
#!/bin/bash
# save as monitor_logins.sh

echo "[*] Monitoring failed login attempts..."
echo "[*] Press Ctrl+C to stop"
echo ""

sudo tail -f /var/log/auth. log | while read line; do
    if echo "$line" | grep -q "Failed password"; then
        timestamp=$(echo "$line" | awk '{print $1, $2, $3}')
        ip=$(echo "$line" | grep -oE '[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+')
        user=$(echo "$line" | grep -oP 'for \K\w+')
        echo "[ALERT] $timestamp - Failed login for '$user' from $ip"
    fi
done
```

Generate test data (new terminal):

```
# Try invalid SSH login
ssh invaliduser@localhost
```

Deliverable: Screenshot of real-time failed login monitoring

Task 2: Schedule Daily Automated Scan with Cron

Objective: Automate security scanning

Steps:

1. Create scan script:

```
#!/bin/bash
# ~/scripts/daily_scan.sh

LOGDIR="/home/kali/scan_logs"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
TARGET="localhost"

mkdir -p "$LOGDIR"

echo "[*] Starting daily scan at $(date)" >> "$LOGDIR/daily_scan.log"

# Quick port scan
nmap -sV -F "$TARGET" > "$LOGDIR/nmap_${TIMESTAMP}.txt" 2>&1

# Check for SUID binaries (priv esc vectors)
find /usr -perm -4000 -type f 2>/dev/null > "$LOGDIR/suid_${TIMESTAMP}.txt"

# Check listening services
ss -tulpn > "$LOGDIR/services_${TIMESTAMP}. txt" 2>&1

echo "[*] Scan complete at $(date)" >> "$LOGDIR/daily_scan. log"
echo "[*] Results saved to $LOGDIR" >> "$LOGDIR/daily_scan.log"
```

2. Make executable:

```
chmod +x ~/scripts/daily_scan. sh
```

3. Test script:

```
~/scripts/daily_scan.sh
cat ~/scan_logs/daily_scan.log
```

4. Schedule with cron:

```
crontab -e

# Add line for daily scan at 3 AM
0 3 * * * /home/kali/scripts/daily_scan. sh >> /home/kali/scan_logs/cron.log 2>&1
```

5. Verify cron entry:

```
crontab -l
```

Deliverable:

- Screenshot of crontab -l showing scheduled task
- Screenshot of scan output files

Task 3: Analyze System Resources Under Stress

Objective: Understand resource monitoring during high load

Steps:

1. Open monitoring terminal:

```
# Terminal 1 - htop or top
htop

# Or multiple monitors
watch -n 1 'free -h; echo "---"; vmstat 1 1'
```

2. Create stress (new terminal):

```
# Install stress tool
sudo apt install stress

# CPU stress (4 workers for 30 seconds)
stress --cpu 4 --timeout 30s

# Memory stress (allocate 1GB)
stress --vm 2 --vm-bytes 512M --timeout 30s

# I/O stress
stress --io 2 --timeout 30s

# Combined stress
stress --cpu 2 --vm 1 --vm-bytes 256M --io 1 --timeout 60s
```

3. Document observations:

```
# During stress, run these:
free -h
vmstat 1 5
iostat -x 1 5
sar -u 1 5
```

4. Create analysis script:

```
#!/bin/bash
# resource_monitor.sh

echo "=== System Resource Analysis ==="
echo "Time: $(date)"
echo ""

echo "=== CPU Info ==="
grep "model name" /proc/cpuinfo | head -1
echo "Load Average: $(uptime | awk -F'load average:' '{print $2}')"
echo ""

echo "=== Memory ==="
free -h
echo ""

echo "=== Disk Usage ==="
df -h /
echo ""

echo "=== Top 5 CPU Processes ==="
ps aux --sort=-%cpu | head -6
echo ""

echo "=== Top 5 Memory Processes ==="
ps aux --sort=-%mem | head -6
```

Deliverable:

- Screenshots of htop during stress test
- Resource analysis report before and during stress

Module 12 Checkpoint

Before moving to Module 13, confirm:

- ☐ Can gather system information with uname, hostname, uptime
- ☐ Can monitor memory with free and vmstat
- ☐ Know important log file locations

- [ ] Can view and filter logs with tail, grep, journalctl
- [ ] Can analyze authentication logs for security events
- [ ] Can create and manage cron jobs
- [ ] Can schedule one-time tasks with at
- [ ] Understand load average and resource metrics

Module 13: Security Fundamentals

Introduction

Security isn't an add-on — it's **built into Linux from the ground up**. Understanding Linux security mechanisms is essential for both defending systems and finding their weaknesses during penetration tests.

This module covers permissions, firewalls, SSH hardening, privilege management, and mandatory access control systems. These are the fundamentals every security professional must master.

13.1 File Permissions Revisited

Permission Deep Dive

Recap from Module 3 with security focus:

```
ls -la /etc/shadow
# -rw-r----- 1 root shadow 1234 Dec 25 10:00 /etc/shadow
#  ||| ||| |||
#  ||| ||| └── Others:  --- (no access)
#  ||| └── Group (shadow): r-- (read only)
#  └── Owner (root): rw- (read + write)
```

Critical Security Permissions

File/Directory	Correct Permission	Reason
`/etc/shadow`	`640` root: shadow	Password hashes
`/etc/passwd`	`644` root:root	User info (world-readable is OK)
`/etc/sudoers`	`440` root:root	Sudo configuration
`~/.ssh/`	`700`	SSH directory
`~/.ssh/id_rsa`	`600`	Private key
`~/.ssh/authorized_keys`	`600`	Authorized keys
`/tmp`	`1777` (sticky)	World-writable temp

Security Permission Checks

```
# Find world-writable files
find / -type f -perm -o+w 2>/dev/null

# Find world-writable directories (without sticky bit)
find / -type d -perm -o+w !  -perm -1000 2>/dev/null

# Find SUID files
find / -type f -perm -4000 2>/dev/null

# Find SGID files
find / -type f -perm -2000 2>/dev/null

# Find files with no owner
find / -nouser -o -nogroup 2>/dev/null

# Find files writable by group
find /etc -type f -perm -g+w 2>/dev/null

# Check sensitive file permissions
stat -c "%a %U:%G %n" /etc/shadow /etc/passwd /etc/sudoers
```

Fixing Insecure Permissions

```
# Lock down SSH directory
chmod 700 ~/. ssh
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub
chmod 600 ~/. ssh/authorized_keys
chmod 644 ~/.ssh/known_hosts

# Lock down sensitive configs
sudo chmod 640 /etc/shadow
sudo chmod 600 /etc/sudoers
sudo chmod 600 /etc/ssh/sshd_config

# Remove world-writable from scripts
find ~/scripts -type f -perm -o+w -exec chmod o-w {} \;

# Set restrictive umask
echo "umask 027" >> ~/.bashrc
# New files:  640, directories: 750
```

13.2 Firewall Basics

UFW (Uncomplicated Firewall)

UFW is the standard firewall interface for Debian/Ubuntu systems — a user-friendly frontend for iptables.

UFW Basic Commands

```
# Check status
sudo ufw status
sudo ufw status verbose
sudo ufw status numbered

# Enable/Disable firewall
sudo ufw enable
sudo ufw disable

# Default policies
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Reset to defaults
sudo ufw reset
```

UFW Rules

```
# Allow by service name
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https

# Allow by port
sudo ufw allow 22
sudo ufw allow 80/tcp
sudo ufw allow 53/udp

# Allow port range
sudo ufw allow 6000:6007/tcp

# Allow from specific IP
sudo ufw allow from 192.168.1.100
sudo ufw allow from 192.168.1.100 to any port 22

# Allow from subnet
sudo ufw allow from 192.168.1.0/24

# Allow to specific interface
sudo ufw allow in on eth0 to any port 80

# Deny rules
sudo ufw deny 23                    # Deny telnet
sudo ufw deny from 10.0.0.5         # Block specific IP
sudo ufw deny from 10.0.0.0/8       # Block subnet

# Reject (sends response vs silent deny)
sudo ufw reject 23

# Delete rules
sudo ufw status numbered
sudo ufw delete 3                   # By number
sudo ufw delete allow 80            # By rule

# Insert rule at position
sudo ufw insert 1 allow from 192.168.1.1
```

UFW Application Profiles

```
# List available profiles
sudo ufw app list

# Show profile details
sudo ufw app info OpenSSH
sudo ufw app info "Apache Full"

# Allow application
sudo ufw allow "OpenSSH"
sudo ufw allow "Apache Full"

# Profiles location
ls /etc/ufw/applications.d/
```

Recommended UFW Setup

```
#!/bin/bash
# Secure UFW configuration

# Reset
sudo ufw --force reset

# Set defaults
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Allow SSH (essential!)
sudo ufw allow 22/tcp

# Rate limit SSH (prevent brute force)
sudo ufw limit ssh

# Allow web traffic (if web server)
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Allow specific management IP
sudo ufw allow from 192.168.1.50 to any port 22

# Enable logging
sudo ufw logging on

# Enable firewall
sudo ufw enable

# Verify
sudo ufw status verbose
```

---

iptables (Advanced)

iptables is the underlying netfilter interface — more powerful but complex.

iptables Structure

```
Tables → Chains → Rules
         │
         ├── INPUT    (incoming to this host)
         ├── OUTPUT   (outgoing from this host)
         ├── FORWARD  (passing through, routing)
         ├── PREROUTING  (before routing decision)
         └── POSTROUTING (after routing decision)
```

Basic iptables Commands

```
# List rules
sudo iptables -L -n -v
sudo iptables -L -n --line-numbers

# List specific table
sudo iptables -t nat -L -n

# Set default policy
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT

# Allow established connections
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Allow loopback
sudo iptables -A INPUT -i lo -j ACCEPT

# Allow SSH
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Allow HTTP/HTTPS
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Block specific IP
sudo iptables -A INPUT -s 10.0.0.5 -j DROP

# Delete rule by number
sudo iptables -D INPUT 3

# Flush all rules
sudo iptables -F

# Save rules (Debian)
sudo iptables-save > /etc/iptables/rules.v4
sudo apt install iptables-persistent

# Restore rules
sudo iptables-restore < /etc/iptables/rules.v4
```

iptables Rate Limiting

```
# Limit SSH connections (prevent brute force)
sudo iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --set
sudo iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --update --seconds 60 --hitcount 4 -j DROP

# Limit ICMP
sudo iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s --limit-burst 4 -j ACCEPT
```

13.3 SSH Hardening

SSH Configuration File

Location: `/etc/ssh/sshd_config`

```
# Edit SSH configuration
sudo nano /etc/ssh/sshd_config
```

Essential SSH Hardening Settings

```
# /etc/ssh/sshd_config

# Change default port (security through obscurity, but helps)
Port 2222

# Disable root login
PermitRootLogin no

# Disable password authentication (keys only)
PasswordAuthentication no
PubkeyAuthentication yes

# Disable empty passwords
PermitEmptyPasswords no

# Limit authentication attempts
MaxAuthTries 3

# Limit sessions
MaxSessions 2

# Set idle timeout
ClientAliveInterval 300
ClientAliveCountMax 2

# Disable X11 forwarding (unless needed)
X11Forwarding no

# Disable TCP forwarding (unless needed)
AllowTcpForwarding no

# Restrict users/groups
AllowUsers kali admin
# OR
AllowGroups sshusers

# Use only SSH protocol 2
Protocol 2

# Stricter key exchange algorithms
KexAlgorithms curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-nistp384

# Disable weak ciphers
Ciphers chacha20-poly1305@openssh. com,aes256-gcm@openssh. com,aes128-gcm@openssh. com

# Strong MACs
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com

# Logging
LogLevel VERBOSE
```

Apply SSH Changes

```
# Validate configuration
sudo sshd -t

# Restart SSH
sudo systemctl restart sshd

# Test connection (in new terminal before closing current!)
ssh -p 2222 user@host
```

SSH Key Security

```
# Generate strong key
ssh-keygen -t ed25519 -a 100 -C "kali@mybox"
# OR RSA with strong bits
ssh-keygen -t rsa -b 4096 -C "kali@mybox"

# Protect private key
chmod 600 ~/.ssh/id_ed25519

# Add passphrase to existing key
ssh-keygen -p -f ~/.ssh/id_ed25519

# Use SSH agent for convenience
eval $(ssh-agent)
ssh-add ~/. ssh/id_ed25519
```

Fail2Ban Integration

```
# Install fail2ban
sudo apt install fail2ban

# Create local config
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local

# Edit config
sudo nano /etc/fail2ban/jail.local
```

```
# /etc/fail2ban/jail.local
[DEFAULT]
bantime = 1h
findtime = 10m
maxretry = 5
banaction = ufw

[sshd]
enabled = true
port = 2222
filter = sshd
logpath = /var/log/auth. log
maxretry = 3
```

```
# Start fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban

# Check status
sudo fail2ban-client status
sudo fail2ban-client status sshd

# Unban IP
sudo fail2ban-client set sshd unbanip 192.168.1.100
```

13.4 User Privilege Management

Principle of Least Privilege

- Users should have **minimum permissions** needed for their tasks
- Use `sudo` for specific commands, not full root access
- Regularly audit user permissions

Sudo Configuration Best Practices

```
# Edit sudoers safely
sudo visudo
```

```
# /etc/sudoers examples

# Full sudo access
kali    ALL=(ALL:ALL) ALL

# Specific commands only
pentester    ALL=(ALL) NOPASSWD:  /usr/bin/nmap, /usr/bin/nikto

# Group-based access
%developers    ALL=(ALL) /usr/bin/docker, /usr/bin/docker-compose

# No password for specific commands
backup_user    ALL=(ALL) NOPASSWD: /usr/bin/rsync, /usr/bin/tar

# Require password for dangerous commands
admin    ALL=(ALL) ALL, !/usr/bin/passwd root, !/usr/bin/su

# Log all sudo commands
Defaults    logfile="/var/log/sudo.log"
Defaults    log_input, log_output
```

Audit User Privileges

```
# Check sudo access for user
sudo -l -U username

# List all sudoers
grep -r "ALL" /etc/sudoers /etc/sudoers.d/

# Find users with shell access
grep -v "nologin\|false" /etc/passwd | cut -d: -f1

# Find users in sudo group
getent group sudo

# Check for NOPASSWD entries
grep -r "NOPASSWD" /etc/sudoers /etc/sudoers.d/

# Review sudo log
sudo cat /var/log/auth.log | grep sudo
```

13.5 File Integrity Monitoring

Using AIDE (Advanced Intrusion Detection Environment)

```
# Install AIDE
sudo apt install aide

# Initialize database
sudo aideinit

# Move database
sudo mv /var/lib/aide/aide.db. new /var/lib/aide/aide.db

# Run check
sudo aide --check

# Update database after legitimate changes
sudo aide --update
sudo mv /var/lib/aide/aide.db.new /var/lib/aide/aide.db
```

Simple File Integrity Checking

```
#!/bin/bash
# Simple integrity checker

MONITOR_DIR="/etc"
HASH_FILE="/var/lib/integrity/hashes.txt"
CHANGES_LOG="/var/log/integrity_changes. log"

mkdir -p /var/lib/integrity

if [ "$1" = "init" ]; then
    # Create baseline
    find "$MONITOR_DIR" -type f -exec sha256sum {} \; > "$HASH_FILE"
    echo "Baseline created:  $(wc -l < "$HASH_FILE") files"
elif [ "$1" = "check" ]; then
    # Check for changes
    echo "=== Integrity Check $(date) ===" >> "$CHANGES_LOG"

    find "$MONITOR_DIR" -type f -exec sha256sum {} \; 2>/dev/null | \
        while read hash file; do
            orig_hash=$(grep "$file" "$HASH_FILE" | awk '{print $1}')
            if [ -z "$orig_hash" ]; then
                echo "NEW FILE: $file" | tee -a "$CHANGES_LOG"
            elif [ "$hash" != "$orig_hash" ]; then
                echo "MODIFIED:  $file" | tee -a "$CHANGES_LOG"
            fi
        done
else
    echo "Usage: $0 {init|check}"
fi
```

Critical File Monitoring

```
# Files to monitor for changes
/etc/passwd
/etc/shadow
/etc/group
/etc/sudoers
/etc/ssh/sshd_config
/etc/hosts
/etc/crontab
/etc/systemd/system/*

# Quick check with stat
for file in /etc/passwd /etc/shadow /etc/sudoers; do
    stat -c "%n:  %y (modified) %z (changed)" "$file"
done
```

13.6 SELinux & AppArmor

AppArmor (Default on Debian/Ubuntu/Kali)

AppArmor uses profiles to restrict program capabilities.

```
# Check status
sudo aa-status

# List profiles
ls /etc/apparmor. d/

# Profile modes:
# - enforce: Actively blocks violations
# - complain: Logs violations but allows
# - disable: Profile not loaded

# Put profile in complain mode
sudo aa-complain /etc/apparmor. d/usr.bin.program

# Put profile in enforce mode
sudo aa-enforce /etc/apparmor.d/usr. bin.program

# Disable profile
sudo ln -s /etc/apparmor. d/usr.bin.program /etc/apparmor. d/disable/
sudo apparmor_parser -R /etc/apparmor.d/usr.bin.program

# Reload all profiles
sudo systemctl reload apparmor
```

SELinux (Default on RHEL/CentOS)

SELinux provides mandatory access control with labels.

```
# Check status
getenforce
sestatus

# Modes:
# - Enforcing: Actively enforces policy
# - Permissive: Logs but doesn't enforce
# - Disabled: SELinux off

# Temporary mode change
sudo setenforce 0    # Permissive
sudo setenforce 1    # Enforcing

# Permanent change:  edit /etc/selinux/config
SELINUX=enforcing

# View file context
ls -Z /etc/passwd

# View process context
ps auxZ | grep httpd

# Restore default context
sudo restorecon -Rv /var/www/html
```

When MAC Causes Issues

```
# Check audit log for denials
# AppArmor
sudo dmesg | grep apparmor

# SELinux
sudo ausearch -m avc -ts recent
sudo sealert -a /var/log/audit/audit.log
```

Key Terms

Term	Definition
Firewall	Network security system controlling incoming/outgoing traffic.
UFW	Uncomplicated Firewall — user-friendly iptables frontend.
iptables	Linux kernel firewall with powerful rule-based filtering.
SSH Hardening	Securing SSH by disabling weak features and enforcing strong auth.
Fail2Ban	Intrusion prevention scanning logs and banning malicious IPs.
Least Privilege	Principle of giving minimum necessary permissions.
MAC	Mandatory Access Control (SELinux, AppArmor) — kernel-enforced security.
DAC	Discretionary Access Control — traditional Unix permissions.

FAQs

Q: Should I use UFW or iptables?

> Use **UFW** for simple setups and ease of management. Learn **iptables** for advanced scenarios, scripting, and understanding how firewalls actually work. UFW generates iptables rules under the hood.

Q: Is changing SSH port from 22 really secure?

> It's "security through obscurity" — doesn't stop determined attackers but reduces noise from automated scanners. Always combine with key-based auth, fail2ban, and proper firewall rules.

Q: How do I recover if I lock myself out with firewall rules?

> Prevention: Always test rules before making permanent, keep a console session open. Recovery: Access via console/KVM, boot into recovery mode, or use cloud provider's console.
>
> ```
> # Before enabling, ensure SSH is allowed
> sudo ufw allow ssh
> sudo ufw enable
> ```

Q: What's the difference between SELinux and AppArmor?

> | Feature | SELinux | AppArmor |
> | --- | --- | --- |
> | Approach | Labels on files/processes | Path-based profiles |
> | Complexity | High | Lower |
> | Default | RHEL/CentOS/Fedora | Debian/Ubuntu/SUSE |
> | Granularity | Very fine | Moderate |

Q: How do I check if my system has security vulnerabilities?

> ```
> # Check for CVEs in installed packages
> sudo apt install debsecan
> debsecan
>
> # Use Lynis for security audit
> sudo apt install lynis
> sudo lynis audit system
> ```

Practice Tasks

Task 1: Configure UFW for Web Server

Objective: Allow only SSH and HTTP/HTTPS traffic

Steps:

```
# Check current status
sudo ufw status

# Reset if needed
sudo ufw --force reset

# Set default policies
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Allow SSH (CRITICAL - do this first!)
sudo ufw allow 22/tcp

# Limit SSH (rate limiting)
sudo ufw limit ssh comment "Rate limit SSH"

# Allow HTTP and HTTPS
sudo ufw allow 80/tcp comment "HTTP"
sudo ufw allow 443/tcp comment "HTTPS"

# Enable firewall
sudo ufw enable

# Verify configuration
sudo ufw status verbose

# Test connectivity
curl -I http://localhost
ssh localhost
```

Expected Output:

```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp                     LIMIT       Anywhere
80/tcp                     ALLOW       Anywhere
443/tcp                    ALLOW       Anywhere
```

Deliverable: Screenshot of `ufw status verbose` output

Task 2: Harden SSH Configuration

Objective: Secure SSH server settings

Steps:

1. Backup original config:

```
sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.backup
```

2. Edit configuration:

```
sudo nano /etc/ssh/sshd_config
```

3. Apply these changes:

```
# Change port
Port 2222

# Disable root login
PermitRootLogin no

# Disable password auth (after setting up keys!)
# PasswordAuthentication no

# Limit attempts
MaxAuthTries 3

# Set timeout
ClientAliveInterval 300
ClientAliveCountMax 2

# Disable X11
X11Forwarding no

# Logging
LogLevel VERBOSE
```

4. Validate and restart:

```
# Check syntax
sudo sshd -t

# Restart SSH
sudo systemctl restart sshd

# Update firewall for new port
sudo ufw allow 2222/tcp
sudo ufw delete allow 22/tcp
```

5. Test connection (NEW TERMINAL!):

```
ssh -p 2222 kali@localhost
```

Deliverable:

- Screenshot of `sshd -t` (no errors)
- Screenshot of successful connection on new port

Task 3: Find World-Writable Files

Objective: Identify potential security risks

Steps:

```
# Find world-writable files (excluding /proc, /sys)
sudo find / -type f -perm -o+w \
    -not -path "/proc/*" \
    -not -path "/sys/*" \
    2>/dev/null | tee /tmp/world_writable.txt

# Count results
echo "Found:  $(wc -l < /tmp/world_writable.txt) world-writable files"

# Find world-writable directories WITHOUT sticky bit
sudo find / -type d -perm -o+w !  -perm -1000 \
    -not -path "/proc/*" \
    -not -path "/sys/*" \
    2>/dev/null | tee /tmp/world_writable_dirs. txt

# Analyze results
echo "=== Potentially Dangerous Files ==="
cat /tmp/world_writable.txt | head -20

# Check for sensitive locations
grep -E "^/(etc|usr|bin|sbin)" /tmp/world_writable.txt

# Security script
#!/bin/bash
echo "=== World-Writable Security Audit ==="

echo -e "\n[*] World-writable files:"
find / -type f -perm -o+w \
    -not -path "/proc/*" -not -path "/sys/*" \
    2>/dev/null | wc -l

echo -e "\n[*] Unsafe world-writable directories:"
find / -type d -perm -o+w ! -perm -1000 \
    -not -path "/proc/*" -not -path "/sys/*" \
    2>/dev/null | wc -l

echo -e "\n[*] SUID files:"
find / -type f -perm -4000 2>/dev/null | wc -l

echo -e "\n[*] Files with no owner:"
find / -nouser 2>/dev/null | wc -l
```

Fix dangerous permissions:

```
# Remove world-writable from specific file
sudo chmod o-w /path/to/file

# Add sticky bit to world-writable directory
sudo chmod +t /path/to/directory
```

Deliverable: List of world-writable files and directories found

---

✅ Module 13 Checkpoint

Before moving to Module 14, confirm:

- ☐ Understand file permission security implications
- ☐ Can configure UFW firewall rules
- ☐ Know basic iptables concepts
- ☐ Can harden SSH configuration
- ☐ Understand SSH key security
- ☐ Can audit user privileges
- ☐ Know basics of AppArmor/SELinux

- [ ] Can find security-risk file permissions

Module 14: Advanced Commands & Tricks

Introduction

This module covers power-user commands and techniques that separate beginners from experts. These tools and tricks enable efficient file finding, data compression, powerful pattern matching, and shell customization.

Master these, and you'll work faster, script better, and handle complex tasks with ease.

14.1 Finding Files

Command: `find`

Search for files in directory hierarchy — most powerful file finder

Syntax:

```
find [PATH] [OPTIONS] [EXPRESSION]
```

Finding by Name

```
 # Exact name
find /home -name "file.txt"

# Case-insensitive
find /home -iname "file.txt"

# Wildcard patterns
find /var/log -name "*.log"
find /etc -name "*.conf"
find .  -name "*backup*"

# Multiple names
find / -name "*. txt" -o -name "*.log"
```

Finding by Type

```
 # Files only
find /etc -type f

# Directories only
find /home -type d

# Symbolic links
find /usr -type l

# Block devices
find /dev -type b

# Character devices
find /dev -type c
```

Finding by Size

```
# Exactly 10MB
find / -size 10M

# Greater than 100MB
find / -size +100M

# Less than 1KB
find /tmp -size -1k

# Range
find /var -size +10M -size -100M

# Size units:  c (bytes), k (KB), M (MB), G (GB)
```

Finding by Time

```
# Modified in last 24 hours
find /var/log -mtime -1

# Modified more than 7 days ago
find /tmp -mtime +7

# Accessed in last 60 minutes
find .  -amin -60

# Changed in last 2 days
find /etc -ctime -2

# Newer than reference file
find .  -newer reference_file.txt

# Modified in last 24 hours (forensics)
find / -type f -mtime 0 2>/dev/null

# Time options:
# -mtime n: modified n*24 hours ago
# -atime n: accessed n*24 hours ago
# -ctime n:  status changed n*24 hours ago
# -mmin n: modified n minutes ago
```

Finding by Permissions

```
# SUID files
find / -perm -4000 -type f 2>/dev/null

# SGID files
find / -perm -2000 -type f 2>/dev/null

# World-writable
find / -perm -o+w -type f 2>/dev/null

# Exact permissions
find /etc -perm 644

# At least these permissions
find /home -perm -644

# Any of these permissions
find /var -perm /666
```

Finding by Owner

```
# By user
find /home -user kali
find / -user root -perm -4000

# By group
find /var -group www-data

# No owner (orphaned)
find / -nouser
find / -nogroup
```

Actions on Found Files

```
# Delete found files
find /tmp -name "*.tmp" -delete

# Execute command on each
find .  -name "*.txt" -exec cat {} \;
find .  -name "*.sh" -exec chmod +x {} \;

# Execute with confirmation
find .  -name "*.bak" -ok rm {} \;

# Execute command with all files at once
find .  -name "*.txt" -exec cat {} +

# Print with custom format
find . -name "*.log" -printf "%p %s %T+\n"
```

Complex Find Examples

```
# Find large files modified recently
find /var -type f -size +50M -mtime -7

# Find and grep
find /etc -name "*.conf" -exec grep -l "password" {} \;

# Find files, exclude directories
find /var -path "/var/cache" -prune -o -name "*. log" -print

# Forensic:  files modified in last hour by root
find / -user root -mmin -60 -type f 2>/dev/null

# Find empty files/directories
find /tmp -empty

# Find broken symlinks
find / -xtype l 2>/dev/null
```

Command: `locate`

Fast file search using database

```
# Install and update database
sudo apt install mlocate
sudo updatedb

# Basic search
locate filename
locate "*. conf"

# Case-insensitive
locate -i FileName

# Limit results
locate -n 10 "*.log"

# Count matches
locate -c "*.txt"

# Match whole path
locate -r "/home/.*\. txt$"
```

> **locate vs find:** `locate` is faster (uses database) but may not find recent files. `find` is real-time but slower.

Command: `which`

Find executable in PATH

```
which python
# /usr/bin/python

which -a python    # All matches in PATH
which nmap bash ls
```

Command: `whereis`

Locate binary, source, and man pages

```
whereis python
# python: /usr/bin/python /usr/lib/python /usr/share/man/man1/python. 1.gz

whereis -b nmap    # Binary only
whereis -m nmap    # Man pages only
whereis -s gcc     # Source only
```

Command: `type`

Describe command type

```
type ls
# ls is aliased to 'ls --color=auto'

type cd
# cd is a shell builtin

type /usr/bin/find
# /usr/bin/find is /usr/bin/find

type -a python    # All locations
```

14.2 Archiving & Compression

Command: `tar`

Tape Archive — create/extract archives

Syntax:

```
tar [OPTIONS] ARCHIVE_NAME FILES
```

Key Options:

Option	Description
`-c`	Create archive
`-x`	Extract archive
`-t`	List contents
`-f FILE`	Archive filename
`-v`	Verbose
`-z`	Gzip compression
`-j`	Bzip2 compression
`-J`	XZ compression
`-C DIR`	Change directory
`-p`	Preserve permissions

Creating Archives

```
 # Create tar (no compression)
tar -cvf archive.tar folder/

# Create with gzip
tar -czvf archive.tar.gz folder/

# Create with bzip2 (better compression, slower)
tar -cjvf archive.tar.bz2 folder/

# Create with xz (best compression, slowest)
tar -cJvf archive.tar.xz folder/

# Multiple items
tar -czvf backup.tar.gz file1.txt folder1/ folder2/

# Exclude files
tar -czvf backup.tar.gz --exclude='*.log' --exclude='.git' folder/

# Timestamped backup
tar -czvf "backup_$(date +%Y%m%d_%H%M%S).tar.gz" folder/
```

Extracting Archives

```
# Extract tar
tar -xvf archive. tar

# Extract gzip
tar -xzvf archive.tar.gz

# Extract to specific directory
tar -xzvf archive.tar.gz -C /target/dir/

# Extract specific files
tar -xzvf archive.tar. gz file1.txt folder/file2.txt

# List contents without extracting
tar -tzvf archive.tar.gz
```

Individual Compression Tools

gzip / gunzip

```
# Compress (replaces original)
gzip file.txt
# Creates file.txt. gz

# Keep original
gzip -k file.txt

# Decompress
gunzip file.txt. gz
gzip -d file. txt.gz

# Compress with level (1-9)
gzip -9 file.txt    # Best compression
gzip -1 file.txt    # Fastest

# View compressed file
zcat file.txt. gz
zless file.txt.gz
```

bzip2 / bunzip2

```
# Compress (better ratio than gzip)
bzip2 file. txt

# Decompress
bunzip2 file.txt.bz2
bzip2 -d file.txt. bz2

# Keep original
bzip2 -k file. txt

# View
bzcat file.txt.bz2
```

xz / unxz

```
# Compress (best ratio)
xz file. txt

# Decompress
unxz file.txt.xz
xz -d file.txt. xz

# Keep original
xz -k file.txt

# Extreme compression
xz -9e file.txt
```

Command: `zip` / `unzip`

Windows-compatible archive format

```
# Create zip
zip archive.zip file1.txt file2.txt
zip -r archive.zip folder/

# With password
zip -e secure.zip file. txt

# Compression level
zip -9 archive.zip folder/

# Extract
unzip archive.zip
unzip archive.zip -d /target/dir/

# List contents
unzip -l archive.zip

# Test integrity
unzip -t archive.zip
```

Compression Comparison

Format	Extension	Speed	Ratio	Command
gzip	.gz	Fast	Good	`tar -czvf`
bzip2	.bz2	Medium	Better	`tar -cjvf`
xz	.xz	Slow	Best	`tar -cJvf`
zip	.zip	Fast	Good	`zip -r`

14.3 Regex Mastery

Basic vs Extended Regex

Feature	Basic (BRE)	Extended (ERE)
One or more	`\+`	`+`

Feature	Basic (BRE)	Extended (ERE)
Zero or one	`\?`	`?`
Alternation	`|`	`|`
Grouping	`\( \)`	`( )`
Command	`grep`	`grep -E` or `egrep`

Essential Regex Patterns

Anchors:

```
^        # Start of line
$        # End of line
\b       # Word boundary
```

Character Classes:

```
.         # Any character
[abc]    # a, b, or c
[^abc]   # Not a, b, or c
[a-z]    # Lowercase letters
[A-Z]    # Uppercase letters
[0-9]    # Digits
\d       # Digit (Perl regex)
\w       # Word character [a-zA-Z0-9_]
\s       # Whitespace
```

Quantifiers:

```
*        # Zero or more
+        # One or more
?        # Zero or one
{n}      # Exactly n
{n,}     # n or more
{n,m}    # Between n and m
```

Practical Regex Examples

```
# Email address
grep -E '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' file.txt

# IP address
grep -E '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' file.txt

# URL
grep -E 'https?://[^ ]+' file. txt

# Phone number (US)
grep -E '\(? [0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-9]{4}' file.txt

# Valid username (3-16 chars, alphanumeric)
grep -E '^[a-zA-Z0-9_]{3,16}$' file.txt

# MAC address
grep -E '([0-9A-Fa-f]{2}: ){5}[0-9A-Fa-f]{2}' file.txt

# Empty lines
grep -E '^$' file. txt

# Lines starting with #
grep -E '^#' file.txt

# Lines NOT starting with #
grep -E '^[^#]' file.txt

# Extract between quotes
grep -oE '"[^"]*"' file. txt

# Password pattern (8+ chars, upper, lower, digit)
grep -E '^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9]).{8,}$' file.txt
```

Regex with sed

```
# Capture groups
echo "John Smith" | sed -E 's/([A-Za-z]+) ([A-Za-z]+)/\2, \1/'
# Output: Smith, John

# Remove HTML tags
sed -E 's/<[^>]+>//g' page.html

# Extract numbers
echo "Price: $42.99" | sed -E 's/.*\$([0-9.]+).*/\1/'
# Output: 42.99

# Remove duplicate lines
sed -E '$! N; /^(.*)\n\1$/! P; D' file.txt
```

Regex with awk

```
# Match pattern
awk '/error/ {print}' log.txt

# Match in specific field
awk -F:  '$7 ~ /bash/ {print $1}' /etc/passwd

# Negated match
awk '$0 !~ /debug/' log.txt

# Complex pattern
awk '/^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+/ {print $1}' access.log
```

14.4 Aliases & Functions

Creating Aliases

```
# Temporary alias (current session)
alias ll='ls -la'
alias cls='clear'
alias update='sudo apt update && sudo apt upgrade -y'

# View all aliases
alias

# Remove alias
unalias ll
```

Permanent Aliases

Add to `~/.bashrc` or `~/.zshrc`:

```
# Navigation
alias ..='cd ..'
alias ... ='cd .. /..'
alias home='cd ~'

# Safety
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Listing
alias ll='ls -la'
alias la='ls -A'
alias l='ls -CF'
alias lh='ls -lah'

# System
alias update='sudo apt update && sudo apt upgrade -y'
alias install='sudo apt install'
alias search='apt search'

# Network
alias myip='curl -s ifconfig.me'
alias ports='netstat -tulpn'
alias listening='sudo lsof -i -P -n | grep LISTEN'

# Git
alias gs='git status'
alias gc='git commit'
alias gp='git push'
alias gl='git log --oneline'

# Pentesting
alias nmap-quick='nmap -sV -sC -O'
alias nmap-full='nmap -sV -sC -O -p-'
alias serve='python3 -m http.server'

# Apply changes
alias reload='source ~/.bashrc'
```

Shell Functions

More powerful than aliases — accept arguments, contain logic:

```
# Add to ~/.bashrc

# Create and enter directory
mkcd() {
    mkdir -p "$1" && cd "$1"
}

# Extract any archive
extract() {
    if [ -f "$1" ]; then
        case $1 in
            *.tar. bz2)   tar xjf "$1"     ;;
            *.tar.gz)    tar xzf "$1"     ;;
            *.tar.xz)    tar xJf "$1"     ;;
            *.bz2)       bunzip2 "$1"     ;;
            *.gz)        gunzip "$1"      ;;
            *.tar)       tar xf "$1"      ;;
            *. tbz2)      tar xjf "$1"     ;;
            *.tgz)       tar xzf "$1"     ;;
            *.zip)       unzip "$1"       ;;
            *.7z)        7z x "$1"        ;;
            *. rar)       unrar x "$1"     ;;
            *)           echo "Unknown format:  $1" ;;
        esac
    else
        echo "File not found: $1"
    fi
}

# Quick HTTP server
serve() {
    local port=${1:-8000}
    echo "Serving on http://$(hostname -I | awk '{print $1}'):$port"
    python3 -m http.server "$port"
}

# Find in files
fif() {
    grep -rnw .  -e "$1"
}

# Quick backup
backup() {
    cp "$1" "${1}. backup. $(date +%Y%m%d_%H%M%S)"
}

# Process search
psg() {
    ps aux | grep -v grep | grep -i "$1"
}
```

14.5 Environment Variables

Viewing Environment

```
# All environment variables
env
printenv

# Specific variable
echo $PATH
echo $HOME
printenv USER

# Search variables
env | grep SHELL
```

Common Environment Variables

Variable	Description
`$PATH`	Executable search paths
`$HOME`	User's home directory
`$USER`	Current username
`$SHELL`	Current shell
`$PWD`	Current directory
`$OLDPWD`	Previous directory
`$EDITOR`	Default text editor
`$LANG`	System language
`$TERM`	Terminal type

Setting Environment Variables

```
# Current session only
export MY_VAR="value"
export PATH="$PATH:/new/path"

# Unset variable
unset MY_VAR
```

Persistent Environment Variables

User-level: Add to `~/.bashrc` or `~/.profile`

```
# ~/. bashrc
export EDITOR="nano"
export VISUAL="nano"
export PATH="$HOME/bin:$HOME/.local/bin:$PATH"
export GOPATH="$HOME/go"
export HISTSIZE=10000
export HISTFILESIZE=20000

# Custom variables
export MY_TOOLS="$HOME/tools"
export WORDLISTS="/usr/share/wordlists"
```

System-level: Add to `/etc/environment` or `/etc/profile. d/*. sh`

```
# /etc/environment
MY_SYSTEM_VAR="value"

# /etc/profile.d/custom.sh
export SYSTEM_PATH="/opt/tools/bin"
```

Profile Files Loading Order

Bash:

```
Login Shell:   /etc/profile → ~/.bash_profile → ~/.bash_login → ~/.profile
Non-Login:     /etc/bash.bashrc → ~/. bashrc
```

Best Practice:

- Put environment variables in `~/.profile` or `~/.bash_profile`
- Put aliases and functions in `~/.bashrc`
- Source `.bashrc` from `.bash_profile`:

```
# ~/.bash_profile
if [ -f ~/.bashrc ]; then
    source ~/.bashrc
fi
```

14.6 History Management

History Commands

```
# View history
history
history 20          # Last 20 commands

# Search history
history | grep nmap
Ctrl+R              # Reverse search (interactive)

# Clear history
history -c          # Clear memory
history -w          # Write to file
cat /dev/null > ~/. bash_history  # Clear file

# Execute from history
!100                # Run command #100
!!                   # Run last command
! nmap               # Run last command starting with "nmap"
! $                  # Last argument of previous command
! *                  # All arguments of previous command
```

History Configuration

```
# ~/. bashrc

# History size
HISTSIZE=10000          # Commands in memory
HISTFILESIZE=20000      # Commands in file

# Ignore duplicates and spaces
HISTCONTROL=ignoreboth   # ignoredups: ignorespace

# Ignore specific commands
HISTIGNORE="ls:cd:pwd: exit: clear"

# Add timestamps
HISTTIMEFORMAT="%Y-%m-%d %H:%M:%S  "

# Append instead of overwrite
shopt -s histappend

# Save after each command
PROMPT_COMMAND="history -a; history -n"

# Multi-line commands as single entry
shopt -s cmdhist
```

Security Considerations

```
# Don't save secrets
export HISTCONTROL=ignorespace
# Now commands starting with space won't be saved
 mysecretcommand --password=secret

# Clear sensitive history
history -d 150      # Delete line 150

# Disable history entirely (for sensitive sessions)
unset HISTFILE
set +o history
```

Key Terms

Term	Definition
Regex	Regular Expression — pattern language for matching text.
Alias	Shortcut name for command or command sequence.
Environment Variable	Named value accessible to all child processes.
Archive	Single file containing multiple files/directories.
Compression	Reducing file size using algorithms (gzip, bzip2, xz).
Glob	Shell wildcard patterns (`*`, `?`, `[]`) — not regex.
History	Record of previously executed commands.

FAQs

Q: What's the difference between `find` and `locate`?

> - `find`: Real-time search, always current, more options, slower
> - `locate`: Uses database, very fast, may miss recent files
>
> Use `locate` for quick searches, `find` for precise or recent file searches.

Q: How do I find files modified in the last 24 hours?

```
find / -type f -mtime 0 2>/dev/null
# -mtime 0 means "less than 24 hours ago"
# -mtime -1 also works (less than 1 day)
```

Q: What's the difference between `.bashrc` and `.bash_profile`?

> - `.bash_profile`: Loaded for **login** shells (SSH, terminal login)
> - `.bashrc`: Loaded for **non-login** interactive shells (new terminal tab)
>
> Best practice: Put everything in `.bashrc` and source it from `.bash_profile`.

Q: How do I make an alias persistent?

> Add it to `~/.bashrc`:
>
> ```
> echo "alias ll='ls -la'" >> ~/. bashrc
> source ~/.bashrc
> ```

Q: Why doesn't my environment variable work in cron?

> Cron runs with minimal environment. Either:
>
> 1. Define variables in crontab: `PATH=/usr/bin:/bin`
> 2. Source profile in script: `source ~/.bashrc`
> 3. Use full paths in scripts

Practice Tasks

Task 1: Find Files Modified in Last 24 Hours (Forensics)

Objective: Practice find for incident response

Steps:

```
 # Create test files with different timestamps
touch -d "2 days ago" old_file.txt
touch recent_file.txt
mkdir test_forensics
touch test_forensics/new_evidence.txt

# Find files modified in last 24 hours
find / -type f -mtime 0 2>/dev/null | head -50

# More specific:  home directory only
find /home -type f -mtime 0 2>/dev/null

# With details
find /home -type f -mtime 0 -exec ls -la {} \; 2>/dev/null

# Save to report
find /home -type f -mtime 0 -printf "%T+ %p\n" 2>/dev/null | sort -r > /tmp/recent_files.txt

# Find by minutes (last hour)
find /var/log -type f -mmin -60 2>/dev/null

# Forensic script
#!/bin/bash
echo "=== Files Modified in Last 24 Hours ==="
echo "Report generated:  $(date)"
echo ""
find / -type f -mtime 0 \
    -not -path "/proc/*" \
    -not -path "/sys/*" \
    -not -path "/run/*" \
    -printf "%T+ %M %u %g %s %p\n" 2>/dev/null | \
    sort -r | head -100
```

Deliverable: List of recently modified files with timestamps

Task 2: Create Timestamped Compressed Backup

Objective: Practice archiving and compression

Steps:

```
 # Create test directory with files
mkdir -p ~/backup_test/{config,data,logs}
echo "config file" > ~/backup_test/config/app.conf
echo "important data" > ~/backup_test/data/info.txt
echo "log entry" > ~/backup_test/logs/app.log

# Create backup script
cat << 'EOF' > ~/scripts/backup.sh
#!/bin/bash

SOURCE="${1: ?Usage: $0 <source_dir>}"
BACKUP_DIR="${2:-/tmp/backups}"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
HOSTNAME=$(hostname)

# Validate source
if [ !  -d "$SOURCE" ]; then
    echo "Error: Directory not found: $SOURCE"
    exit 1
fi

# Create backup directory
mkdir -p "$BACKUP_DIR"

# Create archive name
SOURCE_NAME=$(basename "$SOURCE")
ARCHIVE="${BACKUP_DIR}/${HOSTNAME}_${SOURCE_NAME}_${TIMESTAMP}. tar.gz"

# Create backup
echo "[*] Creating backup: $ARCHIVE"
tar -czvf "$ARCHIVE" -C "$(dirname "$SOURCE")" "$(basename "$SOURCE")"

# Verify
if [ -f "$ARCHIVE" ]; then
    SIZE=$(du -h "$ARCHIVE" | cut -f1)
    echo "[+] Backup created:  $SIZE"

    # Create checksum
    sha256sum "$ARCHIVE" > "${ARCHIVE}.sha256"
    echo "[+] Checksum: ${ARCHIVE}.sha256"

    # List contents
    echo "[*] Archive contents:"
    tar -tzvf "$ARCHIVE"
else
    echo "[-] Backup failed!"
    exit 1
fi
EOF

chmod +x ~/scripts/backup.sh

# Run backup
~/scripts/backup.sh ~/backup_test /tmp/backups

# Verify
ls -la /tmp/backups/
cat /tmp/backups/*. sha256
```

Deliverable: Screenshot showing backup file with timestamp and checksum

---

Task 3: Create Custom Aliases for Common Commands

Objective: Optimize workflow with aliases and functions

Steps:

1. Create alias file:

```
cat << 'EOF' > ~/.bash_aliases
# Navigation
alias ..='cd ..'
alias ... ='cd ../..'
alias ....='cd ../../..'

# Listing
alias ll='ls -lah'
alias la='ls -A'
alias lt='ls -lahtr'  # Sort by time, recent last

# Safety
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# System
alias update='sudo apt update && sudo apt upgrade -y'
alias install='sudo apt install -y'
alias ports='sudo ss -tulpn'
alias myip='curl -s ifconfig.me && echo'

# Pentesting
alias nmap-quick='nmap -sV -sC -T4'
alias nmap-udp='sudo nmap -sU -T4'
alias serve='python3 -m http.server'
alias listener='nc -lvnp'
alias rockyou='cat /usr/share/wordlists/rockyou.txt'

# Useful
alias grep='grep --color=auto'
alias diff='diff --color=auto'
alias h='history'
alias j='jobs -l'
alias path='echo -e ${PATH//:/\\n}'

# Clear and list
alias cls='clear && ls'
EOF
```

2. Create function file:

```
cat << 'EOF' >> ~/.bash_aliases

# Functions
mkcd() { mkdir -p "$1" && cd "$1"; }

extract() {
    if [ -f "$1" ]; then
        case $1 in
            *.tar.bz2) tar xjf "$1" ;;
            *.tar.gz)  tar xzf "$1" ;;
            *.tar.xz)  tar xJf "$1" ;;
            *.tar)     tar xf "$1" ;;
            *.gz)      gunzip "$1" ;;
            *.zip)     unzip "$1" ;;
            *.7z)      7z x "$1" ;;
            *) echo "Unknown: $1" ;;
        esac
    else
        echo "Not found: $1"
    fi
}

# Quick port check
portcheck() {
    nc -zv "$1" "$2" 2>&1
}

# Find in files
fif() {
    grep -rnw . -e "$1" --color=always
}
EOF
```

3. Source in .bashrc:

```
echo '[ -f ~/.bash_aliases ] && source ~/. bash_aliases' >> ~/.bashrc
source ~/.bashrc
```

4. Test aliases:

```
alias           # List all
ll              # Test listing
myip            # Test IP lookup
mkcd test_dir   # Test function
pwd
```

Deliverable: Screenshot of `alias` output showing custom aliases

✅ Module 14 Checkpoint

Before moving to Module 15, confirm:

- ☐ Can find files by name, size, time, permissions with `find`
- ☐ Know the difference between `find`, `locate`, `which`, `whereis`
- ☐ Can create and extract tar archives with various compression
- ☐ Understand basic and extended regular expressions
- ☐ Can create aliases and shell functions
- ☐ Understand environment variables and profile files
- ☐ Can manage shell history

Module 15: Troubleshooting & Best Practices

Introduction

Every Linux user encounters errors, broken scripts, and mysterious system behavior. The difference between a beginner and an expert is knowing **how to systematically diagnose and fix problems**.

This module covers reading error messages, debugging techniques, common pitfalls, performance optimization, backup strategies, and using documentation effectively — essential skills for self-sufficient Linux mastery.

15.1 Reading Error Messages

Anatomy of Error Messages

Error messages typically contain:

1. **Command/Program** that failed
2. **Error type** or code
3. **Description** of what went wrong
4. **Location** (file, line number)

Common Error Patterns

```
 # Permission denied
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
# FIX: Use sudo or check file permissions

# Command not found
$ nmapp
bash: nmapp: command not found
# FIX: Check spelling, install package, update PATH

# No such file or directory
$ cat /nonexistent/file.txt
cat: /nonexistent/file.txt: No such file or directory
# FIX:  Verify path, check spelling, confirm file exists

# File exists
$ mkdir existing_dir
mkdir: cannot create directory 'existing_dir': File exists
# FIX: Use -p flag or different name

# Directory not empty
$ rmdir non_empty_dir
rmdir: failed to remove 'non_empty_dir':  Directory not empty
# FIX: Use rm -r instead

# Device or resource busy
$ umount /mnt/usb
umount:  /mnt/usb: target is busy
# FIX: Close programs using it, use lsof to find culprit

# Read-only file system
$ touch /readonly/file
touch: cannot touch '/readonly/file':  Read-only file system
# FIX:  Remount as read-write or check mount options

# Broken pipe
$ cat huge_file | head -1
# (sometimes shows: cat: write error:  Broken pipe)
# FIX:  Usually harmless, pipe consumer stopped early
```

Exit Codes

```
 # Check exit code of last command
echo $?

# Common exit codes
0   # Success
1   # General error
2   # Misuse of command
126 # Permission problem (not executable)
127 # Command not found
128 # Invalid exit argument
130 # Ctrl+C (SIGINT)
137 # Killed (SIGKILL, 128+9)
139 # Segfault (128+11)
143 # Terminated (SIGTERM, 128+15)
255 # Exit status out of range

# Use in scripts
if command; then
    echo "Success"
else
    echo "Failed with code:  $?"
fi
```

System Logs for Error Investigation

```
 # Recent system errors
journalctl -p err -b

# Specific service errors
journalctl -u ssh --since "10 min ago"

# Kernel errors
dmesg | tail -50
dmesg | grep -i error

# Application logs
tail -f /var/log/syslog
tail -f /var/log/apache2/error.log

# Authentication failures
grep "Failed" /var/log/auth.log
```

15.2 Debugging Scripts

Debug Mode with `set`

```
#!/bin/bash
set -x    # Print each command before executing
set -e    # Exit on first error
set -u    # Treat unset variables as error
set -o pipefail  # Pipeline fails if any command fails

# Combined (recommended for production scripts)
set -euo pipefail

# Disable debug mode
set +x

# Debug specific section
set -x
problematic_code_here
set +x
```

Running Scripts in Debug Mode

```
# Debug entire script
bash -x script.sh

# Very verbose (shows expansion)
bash -xv script.sh

# Check syntax without running
bash -n script.sh

# Trace with line numbers
PS4='+ ${BASH_SOURCE}:${LINENO}:  ' bash -x script.sh
```

Debugging Techniques

```
#!/bin/bash
# Debug helpers

# Print variable values
debug() {
    echo "[DEBUG] $*" >&2
}

# Trace function calls
trace() {
    echo "[TRACE] Entering:  ${FUNCNAME[1]}" >&2
}

# Example usage
my_function() {
    trace
    local input=$1
    debug "Input value: $input"

    # Your code here
    result=$((input * 2))

    debug "Result:  $result"
    echo $result
}

# Add breakpoints (pause for inspection)
read -p "Press Enter to continue..." </dev/tty

# Print stack trace
print_stack() {
    local i
    echo "Stack trace:" >&2
    for ((i=${#FUNCNAME[@]}-1; i>=0; i--)); do
        echo "  ${BASH_SOURCE[$i]}:${BASH_LINENO[$i-1]} in ${FUNCNAME[$i]}()" >&2
    done
}
```

Common Script Bugs

```
 # Bug 1: Unquoted variables with spaces
file="my file.txt"
cat $file       # WRONG:  tries to cat "my" and "file. txt"
cat "$file"     # CORRECT

# Bug 2: Missing space in test
if [$var = "value"]; then   # WRONG: missing spaces
if [ $var = "value" ]; then # CORRECT

# Bug 3: Using = instead of ==
if [ $var = "value" ]   # Works but ambiguous
if [[ $var == "value" ]] # Preferred in bash

# Bug 4: Integer comparison with string operators
if [ "$num" -gt 10 ]    # Correct for integers
if [[ "$num" > "10" ]]  # WRONG: string comparison

# Bug 5: Command substitution in conditions
if [ $(grep pattern file) ]; then  # WRONG
if grep -q pattern file; then       # CORRECT

# Bug 6: Infinite loop with wrong variable
while [ $counter -lt 10 ]; do
    echo $counter
    # Forgot:  ((counter++))
done

# Bug 7: Testing file existence
if [ -f $file ]     # Fails if $file is empty
if [ -f "$file" ]   # Correct
if [[ -f $file ]]   # Also correct (bash)
```

Error Handling Patterns

```
#!/bin/bash
set -euo pipefail

# Trap errors
trap 'error_handler $?  $LINENO' ERR

error_handler() {
    local exit_code=$1
    local line_no=$2
    echo "Error on line $line_no: exit code $exit_code" >&2
    exit $exit_code
}

# Cleanup on exit
cleanup() {
    echo "Cleaning up..."
    rm -f /tmp/script_temp.*
}
trap cleanup EXIT

# Safe command execution
safe_run() {
    if ! "$@"; then
        echo "Command failed:  $*" >&2
        return 1
    fi
}

# Usage
safe_run mkdir /some/directory
```

15.3 Common Mistakes & Fixes

File & Directory Mistakes

Mistake	Problem	Fix
`rm -rf /`	Deletes everything	Use `rm -rf ./dir` (note the dot)
`chmod -R 777 /`	Breaks permissions	Restore from backup
`chown -R user /`	Breaks ownership	Boot recovery mode
Wrong `>` vs `>>`	Overwrites file	Use `>>` to append
`mv file /dev/null`	File gone forever	It's gone
Space in `rm - rf`	Creates file named `-rf`	`rm . /-rf` or `rm -- -rf`

Permission Mistakes

```
# Script won't execute
./script.sh
# bash: ./script.sh: Permission denied

# Fix
chmod +x script. sh
# Or run with interpreter
bash script.sh

# SSH key rejected
ssh user@host
# Permissions 0644 for 'id_rsa' are too open

# Fix
chmod 600 ~/. ssh/id_rsa
chmod 700 ~/.ssh
```

Network Mistakes

```
# Port already in use
python3 -m http.server 80
# OSError: [Errno 98] Address already in use

# Fix:  Find and kill process
sudo lsof -i : 80
sudo kill PID
# Or use different port
python3 -m http.server 8080

# Can't bind to low port
python3 -m http.server 80
# Permission denied

# Fix:  Use sudo or port > 1024
sudo python3 -m http.server 80
python3 -m http.server 8080
```

Disk Space Issues

```
# No space left on device
cp large_file /destination
# cp: error writing '/destination': No space left on device

# Diagnose
df -h                          # Check filesystem usage
du -sh /* 2>/dev/null | sort -rh | head -10  # Find large dirs

# Fix
sudo apt clean                 # Clear apt cache
sudo journalctl --vacuum-size=100M  # Clear old logs
find /var/log -name "*.gz" -delete  # Remove compressed logs
```

15.4 Performance Optimization

Script Optimization

```
# Bad: Reading file in loop
while read line; do
    grep "pattern" file.txt   # Opens file each iteration!
done < input.txt

# Good: Single operation
grep "pattern" file.txt

# Bad: Using cat unnecessarily
cat file | grep pattern    # Useless use of cat

# Good: Direct input
grep pattern file

# Bad: External commands in loop
for i in $(seq 1 1000); do
    result=$(expr $i + 1)
done

# Good: Built-in arithmetic
for ((i=1; i<=1000; i++)); do
    result=$((i + 1))
done

# Bad: Multiple greps
grep "error" file | grep -v "debug" | grep -v "info"

# Good:  Single grep
grep -E "error" file | grep -Ev "(debug|info)"

# Better: Single awk
awk '/error/ && !/debug/ && !/info/' file
```

System Performance Commands

```
# Identify resource hogs
top -b -n 1 | head -15      # Snapshot
htop                        # Interactive

# High CPU processes
ps aux --sort=-%cpu | head -10

# High memory processes
ps aux --sort=-%mem | head -10

# Disk I/O
sudo iotop -o

# Network usage
sudo iftop
sudo nethogs

# Disk read/write
iostat -x 1

# System bottlenecks
vmstat 1 5
```

Process Management

```
# Nice:  Lower CPU priority
nice -n 19 ./heavy_script.sh

# Renice: Adjust running process
renice 10 -p PID

# Limit resources with timeout
timeout 60 ./script.sh          # Kill after 60 seconds

# Limit with ulimit
ulimit -v 1000000               # Limit virtual memory
ulimit -t 60                    # CPU time limit

# Background with low priority
nice -n 19 ./script.sh &
```

15.5 Backup Strategies

3-2-1 Backup Rule

- **3** copies of data
- **2** different storage types
- **1** offsite location

Backup Commands

```
# Simple copy backup
cp -a /source /backup/source_$(date +%Y%m%d)

# Tar backup
tar -czvf backup_$(date +%Y%m%d).tar.gz /source

# Rsync (incremental, efficient)
rsync -av --delete /source/ /backup/source/

# Rsync to remote
rsync -avz -e ssh /source/ user@remote:/backup/

# Rsync with exclusions
rsync -av \
    --exclude='*.log' \
    --exclude='.cache' \
    --exclude='node_modules' \
    /source/ /backup/
```

Automated Backup Script

```
#!/bin/bash
# backup_system.sh

set -euo pipefail

# Configuration
SOURCE="/home/kali"
BACKUP_BASE="/backup"
REMOTE="user@backup-server:/backups"
RETENTION_DAYS=30
LOG_FILE="/var/log/backup.log"

# Functions
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

# Create backup directory
BACKUP_DIR="$BACKUP_BASE/$(date +%Y%m%d_%H%M%S)"
mkdir -p "$BACKUP_DIR"

log "Starting backup to $BACKUP_DIR"

# Local backup
rsync -av \
    --exclude='.cache' \
    --exclude='Downloads' \
    --exclude='*. tmp' \
    "$SOURCE/" "$BACKUP_DIR/" 2>&1 | tee -a "$LOG_FILE"

# Create archive
ARCHIVE="$BACKUP_BASE/backup_$(date +%Y%m%d).tar.gz"
tar -czf "$ARCHIVE" -C "$BACKUP_BASE" "$(basename "$BACKUP_DIR")"
log "Archive created: $ARCHIVE"

# Remote sync (optional)
if [ -n "$REMOTE" ]; then
    rsync -avz "$ARCHIVE" "$REMOTE" && log "Synced to remote"
fi

# Cleanup old backups
find "$BACKUP_BASE" -name "backup_*.tar.gz" -mtime +$RETENTION_DAYS -delete
find "$BACKUP_BASE" -type d -mtime +$RETENTION_DAYS -exec rm -rf {} +

log "Backup complete"
```

Cron Schedule for Backups

```
# Daily backup at 2 AM
0 2 * * * /home/kali/scripts/backup_system. sh >> /var/log/backup_cron.log 2>&1

# Weekly full backup on Sunday
0 3 * * 0 /home/kali/scripts/full_backup.sh

# Monthly archive
0 4 1 * * /home/kali/scripts/monthly_archive.sh
```

15.6 Documentation

Command: `man`

Manual pages — comprehensive documentation

```
# Read manual
man ls
man bash
man 5 passwd    # Section 5 (file formats)

# Navigation
Space       # Page down
b           # Page back
/pattern    # Search forward
n           # Next match
q           # Quit

# Search for keyword in all manuals
man -k keyword
apropos keyword

# Man sections
1  # User commands
2  # System calls
3  # Library functions
4  # Device files
5  # File formats
7  # Miscellaneous
8  # System admin commands
```

Command: `info`

GNU Info documentation

```
info coreutils
info bash

# Navigation
n       # Next node
p       # Previous node
u       # Up
Enter   # Follow link
q       # Quit
```

`--help` Flag

```
# Quick help
ls --help
nmap --help
grep --help 2>&1 | less

# Usually shorter than man pages
```

Command: `tldr`

Simplified, practical examples

```
# Install
sudo apt install tldr
tldr --update

# Use
tldr tar
tldr find
tldr rsync

# Example output:
# tar
# Archiving utility.
#
# - Create an archive from files:
#   tar -cvf archive.tar file1 file2
#
# - Extract an archive:
#   tar -xvf archive.tar
```

Other Documentation Sources

```
# Built-in help for shell builtins
help cd
help test
help [[

# What is this command?
whatis ls
whatis grep

# Full path and type
type -a python

# Package documentation
/usr/share/doc/package-name/

# Online resources
# - https://linux.die.net/man/
# - https://explainshell.com
# - https://tldr.sh
# - https://cheat.sh/command
```

Key Terms

Term	Definition
Exit Code	Numeric return value indicating command success (0) or failure (non-zero).
Debug Mode	Running scripts with tracing enabled (`set -x`).
Trap	Signal handler that executes code on signals/errors.
Incremental Backup	Backup that only copies changed files since last backup.
Man Pages	Traditional Unix documentation accessed via `man` command.
RTFM	"Read The Fine Manual" — reminder to check documentation first.

❓ FAQs

Q: How do I find out why a command failed?

> 1. Check the error message carefully
> 2. Check exit code: `echo $?`
> 3. Check system logs: `journalctl -xe`
> 4. Run with verbose/debug flags: `command -v` or `bash -x`
> 5. Search the error message online

Q: My script works manually but fails in cron. Why?

> Cron has a minimal environment. Solutions:
>
> ```
> # 1. Use full paths
> /usr/bin/python3 instead of python3
>
> # 2. Set PATH in crontab
> PATH=/usr/local/bin:/usr/bin:/bin
>
> # 3. Source profile in script
> source /home/user/.bashrc
>
> # 4. Log output for debugging
> * * * * * /script.sh >> /tmp/cron.log 2>&1
> ```

Q: How do I undo a wrong command?

> - `Ctrl+C` : Stop running command
> - `Ctrl+Z` : Suspend command
> - For file operations: There's no general "undo"
> - Prevention: Use `-i` flag (interactive) or aliases
> - Recovery: Restore from backup or use recovery tools

Q: What's the best way to learn a new command?

> 1. `tldr command` — Quick practical examples
> 2. `command --help` — Brief options overview
> 3. `man command` — Comprehensive documentation
> 4. Practice in a test environment

🎯 Practice Tasks

Task 1: Debug a Failing Script

Objective: Practice systematic debugging

Create buggy script:

```
 cat << 'EOF' > /tmp/buggy_script.sh
#!/bin/bash
# This script has multiple bugs - find and fix them!

echo "Starting backup process"

BACKUP_DIR=/tmp/backups
SOURCE_DIR=/home/kali/documents
DATE=`date +%Y%m%d`

# Bug 1: Directory might not exist
cd $SOURCE_DIR

# Bug 2: Variable used without checking
if [$BACKUP_DIR = "/tmp/backups"]; then
    echo "Using default backup location"
fi

# Bug 3: Unquoted variable with potential spaces
for file in $(ls $SOURCE_DIR); do
    cp $file $BACKUP_DIR/
done

# Bug 4: No error checking
tar -czf $BACKUP_DIR/backup_$DATE.tar.gz $SOURCE_DIR

echo "Backup complete!"
EOF

chmod +x /tmp/buggy_script.sh
```

Debug it:

```
 # Run with debug mode
bash -x /tmp/buggy_script.sh

# Check syntax
bash -n /tmp/buggy_script.sh

# Fix the script
cat << 'EOF' > /tmp/fixed_script.sh
#!/bin/bash
set -euo pipefail

echo "Starting backup process"

BACKUP_DIR="/tmp/backups"
SOURCE_DIR="/home/kali/documents"
DATE=$(date +%Y%m%d)

# Check source exists
if [ ! -d "$SOURCE_DIR" ]; then
    echo "Error: Source directory not found:  $SOURCE_DIR"
    exit 1
fi

# Create backup directory
mkdir -p "$BACKUP_DIR"

# Check variable properly
if [ "$BACKUP_DIR" = "/tmp/backups" ]; then
    echo "Using default backup location"
fi

# Safe file iteration
if [ -d "$SOURCE_DIR" ]; then
    for file in "$SOURCE_DIR"/*; do
        [ -e "$file" ] && cp "$file" "$BACKUP_DIR/"
    done
fi

# Error checking
if tar -czf "$BACKUP_DIR/backup_$DATE.tar.gz" "$SOURCE_DIR" 2>/dev/null; then
    echo "Backup complete!"
else
    echo "Warning:  Tar encountered issues"
fi
EOF
```

Deliverable: Fixed script with explanations of each bug

Task 2: File Recovery Simulation

Objective: Understand recovery concepts (using testdisk/photorec)

```
# Install recovery tools
sudo apt install testdisk

# Create test scenario
mkdir /tmp/recovery_test
cd /tmp/recovery_test
echo "Important data that was deleted" > important.txt
echo "Another critical file" > critical. txt

# Simulate deletion (in real scenario)
rm important.txt critical.txt

# Recovery options:
# 1. Check trash (GUI systems)
ls ~/. local/share/Trash/files/

# 2. Use testdisk for partition recovery
sudo testdisk

# 3. Use photorec for file recovery
sudo photorec

# 4. For ext filesystems, check journal
sudo debugfs /dev/sda1
# debugfs:  lsdel

# Prevention: Install trash-cli
sudo apt install trash-cli
alias rm='trash-put'
```

Deliverable: Document the recovery process attempted

Task 3: Optimize a Slow Script

Objective: Apply performance optimization

Slow script:

```
cat << 'EOF' > /tmp/slow_script.sh
#!/bin/bash
# Inefficient script - optimize it!

# Count lines in all . txt files
total=0
for file in $(find /var/log -name "*.log" 2>/dev/null); do
    count=$(cat $file | wc -l)
    total=$(expr $total + $count)
done
echo "Total lines:  $total"

# Find unique IPs (inefficient)
for line in $(cat /var/log/auth.log 2>/dev/null); do
    echo $line | grep -oE '[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+'
done | sort | uniq -c
EOF
```

Optimized script:

```
cat << 'EOF' > /tmp/fast_script.sh
#!/bin/bash
# Optimized version

# Count lines efficiently - single command
total=$(find /var/log -name "*.log" -exec cat {} + 2>/dev/null | wc -l)
echo "Total lines: $total"

# Or even faster with xargs
total=$(find /var/log -name "*.log" 2>/dev/null | xargs wc -l 2>/dev/null | tail -1 | awk '{print $1}')
echo "Total lines: $total"

# Find unique IPs - single pipeline
grep -oE '([0-9]{1,3}\.){3}[0-9]{1,3}' /var/log/auth.log 2>/dev/null | \
    sort | uniq -c | sort -rn
EOF
```

Compare performance:

```
time bash /tmp/slow_script.sh
time bash /tmp/fast_script.sh
```

Deliverable: Before/after timing comparison

✅ Module 15 Checkpoint

Before moving to Module 16, confirm:

- ☐ Can read and interpret error messages
- ☐ Can debug scripts using set -x and tracing
- ☐ Know common mistakes and how to avoid them
- ☐ Understand basic performance optimization
- ☐ Have a backup strategy
- ☐ Can effectively use man, --help, and tldr

Module 16: FAQs & Real-World Scenarios

🎯 Introduction

This module addresses the most common questions and real-world situations Linux users and security professionals face. These are practical, scenario-based solutions you'll use throughout your career.

16.1 How to Recover Deleted Files?

Immediate Actions

```
# STOP writing to the disk immediately!
# The more you write, the less recoverable data becomes

# Check trash first (GUI systems)
ls ~/.local/share/Trash/files/

# Restore from trash
mv ~/. local/share/Trash/files/myfile.txt ~/

# If using trash-cli
trash-restore
```

ext4 Recovery with extundelete

```
# Install
sudo apt install extundelete

# Unmount the partition first (or boot from live USB)
sudo umount /dev/sdb1

# List deleted files
sudo extundelete /dev/sdb1 --list-deleted

# Recover specific file
sudo extundelete /dev/sdb1 --restore-file path/to/file

# Recover all deleted files
sudo extundelete /dev/sdb1 --restore-all

# Files recovered to RECOVERED_FILES directory
```

File Carving with PhotoRec

```
# Install
sudo apt install testdisk  # Includes photorec

# Run recovery
sudo photorec /dev/sdb

# Interactive menu:
# 1. Select disk
# 2. Select partition
# 3. Select filesystem type
# 4. Choose where to save recovered files
# 5. Start recovery
```

Prevention

```
# Use trash-cli instead of rm
sudo apt install trash-cli

# Add aliases
alias rm='trash-put'
alias rmdir='trash-put'

# Real rm when needed
\rm file.txt  # Bypass alias

# Or use rm with confirmation
alias rm='rm -i'
```

16.2 How to Kill Unresponsive Processes?

Identify the Process

```
# Find by name
pgrep -l firefox
pidof firefox

# Find with ps
ps aux | grep firefox

# Find with top/htop
top    # Then press 'k' to kill
htop   # Select and press F9
```

Kill Methods (Escalating Force)

```
# 1. Graceful termination (SIGTERM, default)
kill PID
kill -15 PID

# 2. Interrupt (like Ctrl+C)
kill -2 PID
kill -INT PID

# 3. Force kill (SIGKILL - cannot be ignored)
kill -9 PID
kill -KILL PID

# 4. Kill by name
killall firefox
pkill firefox

# 5. Kill all by pattern
pkill -f "python script.py"

# 6. Kill process group
kill -9 -PGID
```

When Nothing Works

```
# Check if process is zombie
ps aux | grep "^Z"
# Zombie processes are already dead - kill parent instead

# Find and kill parent
ps -o ppid= -p ZOMBIE_PID
kill PARENT_PID

# Nuclear option:  reboot
sudo reboot

# Or SysRq magic keys (if system is frozen)
# Alt+SysRq+REISUB (graceful reboot)
```

16.3 What if Permission Denied?

Diagnose the Problem

```
# Check file permissions
ls -la /path/to/file

# Check if you're the owner
stat /path/to/file

# Check your groups
id
groups

# Check if mounted read-only
mount | grep "path"
```

Solutions

```
# 1. Use sudo (if authorized)
sudo cat /etc/shadow

# 2. Change ownership (if you own parent or are root)
sudo chown $USER:$USER file.txt

# 3. Change permissions
chmod u+rwx file.txt
chmod 755 script.sh

# 4. Add yourself to group
sudo usermod -aG group_name $USER
# Then logout/login or:  newgrp group_name

# 5. Check for immutable attribute
lsattr file.txt
# If shows 'i' flag:
sudo chattr -i file.txt

# 6. Check SELinux/AppArmor
getenforce        # SELinux
aa-status         # AppArmor

# 7. Remount read-write (if read-only)
sudo mount -o remount,rw /mount/point
```

Special Cases

```
# Execute permission for scripts
chmod +x script.sh
# Or run with interpreter
bash script.sh
python3 script.py

# Permission denied for network port < 1024
sudo python3 -m http.server 80
# Or use authbind/setcap
sudo setcap 'cap_net_bind_service=+ep' /usr/bin/python3

# SSH key permission errors
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub
chmod 700 ~/.ssh
```

16.4 How to Automate Tasks?

Cron Jobs (Scheduled Tasks)

```
# Edit crontab
crontab -e

# Examples
# Every day at 2 AM
0 2 * * * /home/kali/scripts/backup. sh

# Every hour
0 * * * * /home/kali/scripts/check. sh

# Every 15 minutes
*/15 * * * * /home/kali/scripts/monitor.sh

# Monday-Friday at 9 AM
0 9 * * 1-5 /home/kali/scripts/work.sh

# List cron jobs
crontab -l
```

Systemd Timers (Modern Alternative)

```
# Create service file
sudo nano /etc/systemd/system/mybackup.service
```

```
[Unit]
Description=My Backup Service

[Service]
ExecStart=/home/kali/scripts/backup.sh
User=kali
```

```
# Create timer file
sudo nano /etc/systemd/system/mybackup. timer
```

```
[Unit]
Description=Run backup daily

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
```

```
# Enable and start
sudo systemctl enable mybackup. timer
sudo systemctl start mybackup. timer

# Check status
systemctl list-timers
```

Automation Scripts

```
#!/bin/bash
# recon_automation.sh - Automated reconnaissance

TARGET=$1
OUTPUT_DIR="./recon/$TARGET"
mkdir -p "$OUTPUT_DIR"

echo "[*] Starting recon for $TARGET"

# DNS enumeration
echo "[*] DNS lookup..."
dig "$TARGET" +short > "$OUTPUT_DIR/dns.txt"

# Subdomain enumeration
echo "[*] Checking subdomains..."
for sub in www mail ftp admin api; do
    ip=$(dig +short "$sub.$TARGET")
    [ -n "$ip" ] && echo "$sub.$TARGET: $ip" >> "$OUTPUT_DIR/subdomains. txt"
done

# Port scan
echo "[*] Port scanning..."
nmap -sV -F "$TARGET" -oN "$OUTPUT_DIR/nmap.txt"

echo "[+] Results saved to $OUTPUT_DIR"
```

16.5 How to Secure a Linux Server?

Essential Security Checklist

```
#!/bin/bash
# server_hardening.sh

# 1. Update system
sudo apt update && sudo apt upgrade -y

# 2. Configure firewall
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 22/tcp    # SSH
sudo ufw enable

# 3. Harden SSH
sudo sed -i 's/#PermitRootLogin yes/PermitRootLogin no/' /etc/ssh/sshd_config
sudo sed -i 's/#PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/sshd_config
sudo systemctl restart sshd

# 4. Install fail2ban
sudo apt install fail2ban -y
sudo systemctl enable fail2ban
sudo systemctl start fail2ban

# 5. Configure automatic updates
sudo apt install unattended-upgrades -y
sudo dpkg-reconfigure -plow unattended-upgrades

# 6. Set password policies
sudo apt install libpam-pwquality -y

# 7. Disable unused services
sudo systemctl disable bluetooth
sudo systemctl disable cups

# 8. Configure logging
sudo apt install auditd -y
sudo systemctl enable auditd

# 9. Set file permissions
chmod 700 ~/. ssh
chmod 600 ~/. ssh/authorized_keys

# 10. Create non-root user
# sudo adduser admin
# sudo usermod -aG sudo admin
```

Security Monitoring

```
# Check for rootkits
sudo apt install rkhunter chkrootkit -y
sudo rkhunter --check
sudo chkrootkit

# Audit system
sudo apt install lynis -y
sudo lynis audit system

# Monitor logs
sudo tail -f /var/log/auth.log
journalctl -f -p warning
```

16.6 How to Dual-Boot Safely?

Preparation

```
# 1. Backup important data FIRST!

# 2. Create Windows recovery drive
# 3. Disable Fast Startup in Windows
#    Control Panel → Power Options → Choose what power buttons do
#    → Change settings currently unavailable
#    → Uncheck "Turn on fast startup"

# 4. Disable Secure Boot in BIOS (if needed)
# 5.  Shrink Windows partition from Windows Disk Management
```

Installation Order

1. **Install Windows first** (if fresh install)
2. **Install Linux second** (detects Windows, creates GRUB)

Partition Layout Example

```
/dev/sda1 - EFI System Partition (500MB) - FAT32
/dev/sda2 - Windows Recovery (varies)
/dev/sda3 - Windows System (C:) - NTFS
/dev/sda4 - Linux root (/) - ext4 (50-100GB)
/dev/sda5 - Linux swap (4-8GB)
/dev/sda6 - Linux home (/home) - ext4 (remaining)
```

Boot Management

```
# Update GRUB after changes
sudo update-grub

# Set default OS (0=first entry, 1=second, etc.)
sudo nano /etc/default/grub
# GRUB_DEFAULT=0
# GRUB_TIMEOUT=10
sudo update-grub

# Fix GRUB if broken
# Boot from Linux live USB:
sudo mount /dev/sda4 /mnt
sudo mount /dev/sda1 /mnt/boot/efi
sudo mount --bind /dev /mnt/dev
sudo mount --bind /proc /mnt/proc
sudo mount --bind /sys /mnt/sys
sudo chroot /mnt
grub-install /dev/sda
update-grub
exit
sudo reboot
```

□ Practice Tasks

Task 1: Simulate and Troubleshoot Disk-Full Scenario

Objective: Handle disk space emergencies

```
# Create disk-full simulation (in a safe way)
dd if=/dev/zero of=/tmp/bigfile bs=1M count=1000

# Check disk usage
df -h

# Find space hogs
du -sh /* 2>/dev/null | sort -rh | head -10
du -sh /var/* 2>/dev/null | sort -rh | head -10

# Clear the simulated file
rm /tmp/bigfile

# Real cleanup procedures
sudo apt clean                          # Clear apt cache
sudo journalctl --vacuum-size=100M      # Limit journal size
find /var/log -name "*.gz" -delete      # Remove old logs
find /tmp -type f -atime +7 -delete     # Old temp files

# Find large files
find / -type f -size +100M -exec ls -lh {} \; 2>/dev/null
```

Deliverable: Document cleanup steps and space recovered

Task 2: Controlled Privilege Escalation Practice

Objective: Understand privilege escalation vectors

```
# Create a practice environment (as root)
sudo su

# Create vulnerable SUID binary (for education only!)
cp /bin/bash /tmp/vuln_bash
chmod u+s /tmp/vuln_bash

# Switch to regular user and test
su - kali
ls -la /tmp/vuln_bash
# -rwsr-xr-x 1 root root ...  /tmp/vuln_bash

# Exploit it
/tmp/vuln_bash -p
whoami
# root

# Clean up
exit
sudo rm /tmp/vuln_bash

# Find real SUID binaries
find / -perm -4000 -type f 2>/dev/null

# Check GTFOBins for exploitation methods
# https://gtfobins.github.io/
```

Deliverable: List of SUID binaries found and potential risks

Task 3: Automate Recon with Shell Script

Objective: Create practical automation

```
#!/bin/bash
# automated_recon.sh
```

```
# automated_recon.sh

set -euo pipefail

# Colors
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'

usage() {
    echo "Usage: $0 <target_domain>"
    echo "Example: $0 example.com"
    exit 1
}

[ $# -lt 1 ] && usage

TARGET=$1
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
OUTPUT_DIR="./recon/${TARGET}_${TIMESTAMP}"

mkdir -p "$OUTPUT_DIR"

echo -e "${GREEN}[*] Starting recon for:  $TARGET${NC}"
echo "[*] Output directory: $OUTPUT_DIR"

# DNS Records
echo -e "${YELLOW}[*] Gathering DNS records...${NC}"
{
    echo "=== A Records ==="
    dig +short A "$TARGET"
    echo ""
    echo "=== NS Records ==="
    dig +short NS "$TARGET"
    echo ""
    echo "=== MX Records ==="
    dig +short MX "$TARGET"
    echo ""
    echo "=== TXT Records ==="
    dig +short TXT "$TARGET"
} > "$OUTPUT_DIR/dns_records.txt"

# Subdomains
echo -e "${YELLOW}[*] Checking common subdomains...${NC}"
for sub in www mail ftp ssh admin api dev staging test blog shop; do
    result=$(dig +short "$sub.$TARGET" 2>/dev/null)
    if [ -n "$result" ]; then
        echo -e "${GREEN}[+] Found: $sub.$TARGET${NC}"
        echo "$sub.$TARGET -> $result" >> "$OUTPUT_DIR/subdomains. txt"
    fi
done

# WHOIS
echo -e "${YELLOW}[*] WHOIS lookup...${NC}"
whois "$TARGET" > "$OUTPUT_DIR/whois.txt" 2>/dev/null || echo "WHOIS failed"

# Quick port scan (if nmap available)
if command -v nmap &> /dev/null; then
    echo -e "${YELLOW}[*] Quick port scan...${NC}"
    nmap -sV -F "$TARGET" -oN "$OUTPUT_DIR/nmap_quick.txt" 2>/dev/null
fi

# Summary
echo ""
echo -e "${GREEN}=== Recon Complete ===${NC}"
```

```
echo "Results saved to: $OUTPUT_DIR"
ls -la "$OUTPUT_DIR"
```

Deliverable: Working recon script with output samples

✅ Module 16 Checkpoint

Before moving to Module 17, confirm:

- ☐ Know how to attempt file recovery
- ☐ Can kill unresponsive processes appropriately
- ☐ Understand permission troubleshooting
- ☐ Can set up automation with cron or scripts
- ☐ Know basic server hardening steps
- ☐ Understand dual-boot considerations

Module 17: Master Cheat Sheet

🎯 Introduction

This is your **quick reference guide** — organized one-liners, essential flags, and critical commands grouped by category. Print it, bookmark it, and memorize the key commands.

17.1 Navigation & File Operations

```
 # Navigation
pwd                     # Current directory
cd /path/to/dir         # Change directory
cd ~                    # Home directory
cd -                    # Previous directory
cd ..                   # Parent directory

# Listing
ls -la                  # Long list with hidden files
ls -lah                 # Human-readable sizes
ls -lt                  # Sort by time (newest first)
ls -lS                  # Sort by size (largest first)
tree -L 2               # Tree view, 2 levels deep

# File Operations
touch file.txt          # Create/update file
mkdir -p dir/subdir     # Create nested directories
cp -r source dest       # Copy directory recursively
mv old new              # Move/rename
rm -rf dir              # Force remove directory
ln -s target link       # Create symbolic link

# Viewing Files
cat file                # Display entire file
head -20 file           # First 20 lines
tail -20 file           # Last 20 lines
tail -f file            # Follow file (real-time)
less file               # Paginated view
```

17.2 Search & Find

```
# Find files
find / -name "*.conf"                    # By name
find / -type f -size +100M               # Large files
find / -mtime -1                         # Modified in last 24h
find / -perm -4000 -type f               # SUID files
find / -user root -perm -4000            # Root SUID files
find / -writable -type f                 # Writable files

# Search in files
grep -r "pattern" /path                  # Recursive grep
grep -i "pattern" file                   # Case insensitive
grep -v "pattern" file                   # Invert match
grep -E "regex" file                     # Extended regex
grep -l "pattern" *.txt                  # Files containing pattern
grep -n "pattern" file                   # With line numbers
grep -C 3 "pattern" file                 # 3 lines context

# Quick search
locate filename                          # Fast database search
which command                            # Find command in PATH
whereis command                          # Binary, source, man
```

17.3 Text Processing

```
# Basic Processing
cat file1 file2 > combined               # Concatenate
sort file                                # Sort lines
sort -u file                             # Sort unique
uniq file                                # Remove adjacent duplicates
wc -l file                               # Line count
wc -w file                               # Word count

# Cut & Extract
cut -d: -f1 /etc/passwd                  # Extract field
cut -c1-10 file                          # Extract characters
awk '{print $1}' file                    # Print first column
awk -F:  '{print $1}' /etc/passwd        # Custom delimiter

# Transform
tr 'a-z' 'A-Z'                           # Uppercase
tr -d '\r'                               # Remove carriage returns
sed 's/old/new/g' file                   # Replace all
sed -i 's/old/new/g' file                # In-place replace
sed '/pattern/d' file                    # Delete matching lines
sed -n '10,20p' file                     # Print lines 10-20
```

17.4 User & Permissions

```
# User Management
whoami                                  # Current user
id                                      # User ID and groups
sudo -l                                 # List sudo permissions
su - user                               # Switch user
sudo useradd -m user                    # Add user
sudo passwd user                        # Set password
sudo usermod -aG sudo user              # Add to sudo group
sudo userdel -r user                    # Delete user + home

# Permissions
chmod 755 file                          # rwxr-xr-x
chmod u+x file                          # Add execute for user
chmod -R 644 dir                        # Recursive
chown user:group file                   # Change owner
chown -R user:group dir                 # Recursive

# Special Permissions
chmod u+s file                          # SUID
chmod g+s dir                           # SGID
chmod +t dir                            # Sticky bit
```

17.5 Process Management

```
# View Processes
ps aux                                  # All processes
ps aux | grep process                   # Find process
pgrep -l name                           # Find by name
top                                     # Interactive monitor
htop                                    # Better interactive

# Control Processes
kill PID                                # Graceful kill (SIGTERM)
kill -9 PID                             # Force kill (SIGKILL)
killall name                            # Kill by name
pkill -f pattern                        # Kill by pattern

# Job Control
command &                               # Run in background
jobs                                    # List jobs
fg %1                                   # Bring to foreground
bg %1                                   # Resume in background
Ctrl+Z                                  # Suspend process
nohup command &                         # Survive logout
```

17.6 Networking

```
# Interface Info
ip addr                                 # IP addresses
ip route                                # Routing table
ss -tulpn                               # Listening ports
netstat -tulpn                          # Alternative

# Connectivity
ping -c 4 host                          # Test connectivity
traceroute host                         # Trace path
curl -I url                             # HTTP headers
wget url                                # Download file

# DNS
dig domain                              # DNS lookup
dig +short domain                       # Short answer
nslookup domain                         # Alternative
host domain                             # Simple lookup

# SSH
ssh user@host                           # Connect
ssh -p 2222 user@host                   # Custom port
scp file user@host:/path                # Copy file
scp -r dir user@host:/path              # Copy directory
ssh-keygen -t ed25519                   # Generate key
ssh-copy-id user@host                   # Copy key to server
```

17.7 Package Management (Debian/Kali)

```
sudo apt update                         # Update package list
sudo apt upgrade                        # Upgrade packages
sudo apt install package                # Install
sudo apt remove package                 # Remove
sudo apt purge package                  # Remove + configs
sudo apt autoremove                     # Remove orphans
sudo apt search keyword                 # Search packages
apt show package                        # Package info
dpkg -l                                 # List installed
dpkg -L package                         # List package files
dpkg -S /path/to/file                   # Find package owner
```

17.8 Disk & Storage

```
# Disk Info
df -h                                   # Filesystem usage
du -sh dir                              # Directory size
du -h --max-depth=1 /var                # Subdirectory sizes
lsblk                                   # Block devices
fdisk -l                                # Partition info
blkid                                   # UUID/filesystem

# Mount
mount                                   # List mounts
sudo mount /dev/sdb1 /mnt               # Mount device
sudo umount /mnt                        # Unmount
cat /etc/fstab                          # Permanent mounts
```

17.9 Compression & Archives

```
 # Tar
tar -czvf archive.tar.gz dir/           # Create gzip archive
tar -cjvf archive.tar.bz2 dir/          # Create bzip2 archive
tar -xzvf archive.tar. gz                # Extract gzip
tar -xjvf archive.tar. bz2               # Extract bzip2
tar -tvf archive.tar                    # List contents

# Other
gzip file                               # Compress
gunzip file.gz                          # Decompress
zip -r archive.zip dir/                 # Create zip
unzip archive.zip                       # Extract zip
```

17.10 CyberSec-Specific Commands

Reconnaissance

```
 # DNS Enumeration
dig axfr @ns.domain.com domain.com      # Zone transfer
dig +short domain.com ANY               # All records
host -t mx domain.com                   # Mail servers

# Network Scanning
nmap -sV -sC target                     # Version + scripts
nmap -p- target                         # All ports
nmap -sU -F target                      # UDP scan
nmap -sn 192.168.1.0/24                 # Host discovery
masscan -p1-65535 target --rate=1000    # Fast port scan

# Web Enumeration
nikto -h http://target                  # Web scanner
gobuster dir -u http://target -w wordlist  # Dir brute
ffuf -u http://target/FUZZ -w wordlist  # Fast fuzzer
whatweb http://target                   # Web fingerprint
curl -I http://target                   # Headers
```

Exploitation Support

```
 # Listeners
nc -lvnp 4444                           # Netcat listener
rlwrap nc -lvnp 4444                    # With readline

# File Transfers
python3 -m http.server 8080             # HTTP server
nc -lvnp 1234 > file                    # Receive file
nc target 1234 < file                   # Send file

# Shell Upgrades
python3 -c 'import pty; pty. spawn("/bin/bash")'
export TERM=xterm
Ctrl+Z
stty raw -echo; fg
```

Post-Exploitation

```
# System Enumeration
uname -a                                # Kernel version
cat /etc/os-release                     # OS info
hostname                                # Hostname
whoami && id                            # Current user

# User Enumeration
cat /etc/passwd                         # All users
cat /etc/shadow                         # Password hashes (root)
sudo -l                                 # Sudo permissions
cat /etc/sudoers                        # Sudoers file

# Network Enumeration
ip a                                    # Interfaces
ss -tulpn                               # Listening ports
arp -a                                  # ARP table
cat /etc/hosts                          # Hosts file

# Privilege Escalation Vectors
find / -perm -4000 -type f 2>/dev/null  # SUID binaries
find / -writable -type f 2>/dev/null    # Writable files
cat /etc/crontab                        # Cron jobs
ls -la /etc/cron.*                      # Cron directories
ps aux | grep root                      # Root processes
env                                     # Environment variables
```

Log Analysis

```
# Authentication
grep "Failed password" /var/log/auth.log
grep "Accepted" /var/log/auth.log
journalctl -u ssh --since "1 hour ago"

# System
tail -f /var/log/syslog
journalctl -f -p err
dmesg | tail -50
```

17.11 Essential Flags Quick Reference

Flag	Common Meaning
`-a`	All (ls -a, grep -a)
`-f`	Force (rm -f, cp -f)
`-h`	Human-readable (ls -h, du -h)
`-i`	Interactive/case-insensitive
`-l`	Long format / list
`-n`	Numeric / count
`-o`	Output file
`-p`	Preserve / port
`-q`	Quiet

Flag	Common Meaning
`-r, -R`	Recursive
`-s`	Silent / size
`-v`	Verbose
`-w`	Writable / width
`-x`	Execute / extract

17.12 Emergency Commands

```
# System Frozen
Alt+SysRq+REISUB                        # Safe reboot

# Can't Login
# Boot to recovery mode
# Select root shell
passwd username                          # Reset password

# Disk Full
du -sh /* 2>/dev/null | sort -rh | head -10
sudo apt clean
sudo journalctl --vacuum-size=100M

# Process Consuming Resources
top                                     # Find PID
kill -9 PID                             # Force kill

# Network Not Working
ip link set eth0 up                     # Bring up interface
sudo dhclient eth0                      # Get IP via DHCP
sudo systemctl restart NetworkManager

# SSH Locked Out
# Use console access
sudo nano /etc/ssh/sshd_config
# Fix configuration
sudo systemctl restart sshd
```

Practice Tasks

Task 1: Create Your Personal Cheat Sheet

Objective: Build a customized reference document

```
# Create cheat sheet file
mkdir -p ~/notes
cat << 'EOF' > ~/notes/my_cheatsheet.md
# My Linux Cheat Sheet

## Most Used Commands
- `ll` - List files (alias for ls -la)
- `..` - Go up directory (alias for cd ..)

## Pentesting
### Quick Scans
```

nmap -sV -sC -T4 target

```
### Listeners
```

nc -lvnp 4444

```
## Troubleshooting
### Check Disk Space
```

df -h du -sh /*

```
### Find Large Files
```

find / -type f -size +100M 2>/dev/null

```
## Daily Shortcuts
(Add your own!)

EOF

# View and edit
nano ~/notes/my_cheatsheet. md
```

Deliverable: Your personalized cheat sheet file

Task 2: Memorize 10 Critical Pentesting Commands

Objective: Commit essential commands to memory

The 10 Must-Know Commands:

```
 # 1. Find SUID binaries (privilege escalation)
find / -perm -4000 -type f 2>/dev/null

# 2. Quick port scan
nmap -sV -sC -T4 target

# 3. Start listener
nc -lvnp 4444

# 4. Start HTTP server
python3 -m http.server 8080

# 5. Upgrade shell
python3 -c 'import pty; pty. spawn("/bin/bash")'

# 6. Check sudo permissions
sudo -l

# 7. Extract IPs from file
grep -oE '([0-9]{1,3}\. ){3}[0-9]{1,3}' file

# 8. Find recently modified files
find / -mtime -1 -type f 2>/dev/null

# 9. Check listening ports
ss -tulpn

# 10. View auth logs
grep "Failed\|Accepted" /var/log/auth.log
```

Memorization Exercise:

- Write each command 3 times
- Explain what it does
- Run each one on your system

Task 3: Build Incident Response Quick Reference

Objective: Create ready-to-use IR card

```
cat << 'EOF' > ~/notes/incident_response. md
# Incident Response Quick Reference

## Initial Triage

### 1. Capture Volatile Data (in order)
```bash
date
who
w
ps aux
netstat -tulpn
ss -tulpn
lsof -i
```

2. Network Connections

```
ss -tulpn > /evidence/network_$(date +%s).txt
netstat -an > /evidence/netstat_$(date +%s).txt
arp -a > /evidence/arp_$(date +%s).txt
```

3. Running Processes

```
ps auxf > /evidence/processes_$(date +%s).txt
pstree -p > /evidence/pstree_$(date +%s).txt
```

4. User Activity

```
w > /evidence/users_$(date +%s).txt
last > /evidence/last_$(date +%s).txt
cat /var/log/auth. log > /evidence/auth_$(date +%s).txt
```

5. Persistence Mechanisms

```
crontab -l
ls -la /etc/cron.*
cat /etc/passwd
cat /etc/shadow
find / -perm -4000 2>/dev/null
```

6. Recent File Changes

```
find / -mtime -1 -type f 2>/dev/null > /evidence/recent_files. txt
find / -ctime -1 -type f 2>/dev/null > /evidence/changed_files.txt
```

Evidence Collection
```

```
# Create evidence directory
mkdir -p /evidence

# Hash all evidence
find /evidence -type f -exec sha256sum {} \; > /evidence/hashes.txt

# Create memory dump (if possible)
sudo dd if=/dev/mem of=/evidence/memory. dump bs=1M

# Create disk image
sudo dd if=/dev/sda of=/evidence/disk. img bs=4M status=progress
```

Common IOC Locations

- /tmp (world-writable)
- /dev/shm (tmpfs, often missed)
- /var/tmp (persistent temp)
- ~/. ssh/authorized_keys
- /etc/cron.d/
- /etc/systemd/system/

EOF

Deliverable: Completed IR quick reference document

✅ Module 17 Checkpoint — Course Complete!

Congratulations! You've completed the Linux Mastery Course.

Final Checklist

- ☐ Can navigate filesystem efficiently
- ☐ Master file and text processing
- ☐ Understand and manage permissions
- ☐ Can manage users, groups, and packages
- ☐ Competent with networking commands
- ☐ Can write bash scripts for automation
- ☐ Know security fundamentals
- ☐ Can troubleshoot common issues
- ☐ Have personal cheat sheet ready
- ☐ Critical commands memorized

🎓 Course Completion

What's Next?

1. **Practice Daily** — Use Linux as your primary OS
2. **Build Projects** — Automate real tasks
3. **CTF Challenges** — TryHackMe, HackTheBox, OverTheWire
4. **Certifications** — LPIC-1, CompTIA Linux+, OSCP
5. **Specialize** — Choose your path (pentesting, admin, DevOps)

Resources

- **Practice:** OverTheWire Bandit
- **CTF:** TryHackMe, HackTheBox
- **Reference:** Explainshell, TLDR Pages
- **Security:** GTFOBins, PayloadsAllTheThings

🎉 Course Complete!

Linux Mastery Course by Loki & Shelly

You've covered 17 modules, from foundation to mastery. The commands and concepts in this course form the backbone of Linux administration and cybersecurity operations.

Remember: The best way to learn is to practice. Break things, fix them, automate everything, and never stop exploring.

Happy Hacking!