

Linux & PowerShell Professional Mastery

30-Day Complete Handbook

Master the Command Line. Understand Your Systems. Secure Your Network. Automate Your Work.

30 Days: From Beginner to Advanced IT Professional
Ubuntu 24.04 LTS · Windows 11 · Windows PowerShell 5.1 · PowerShell 7.x

Edition 2026.11.08 · Days 01–02 of 30 · Built 2026-11-08

Copyright © 2026 Linux & PowerShell Professional Mastery contributors. Released under the MIT License.

Contents

1. The 30-Day Course 3
2. Getting Started 6
3. Windows 11 Setup 8
4. Ubuntu 24.04 LTS Setup 10
5. WSL2 Installation 12
6. PowerShell 7 Installation 14
7. Lab Environment & Safety Rules 18
8. Day 01: Introduction to Linux and PowerShell 20
9. Day 02: File and Directory Management 42
10. Path A — Cybersecurity / SOC Analyst 64
11. Path B — Network Analyst / Network Engineer 66
12. Path C — System Administrator / General IT Professional 68

The 30-Day Course

Six modules of five days each. Each day takes about **3–4 hours**: reading, demonstrations, a hands-on lab, a quiz and homework.

> **Release cadence**
>
> Lessons are published **two days at a time, every night at 22:00 IST (16:30 UTC)**. Unreleased days appear below without links. See the release schedule for exact dates.

Module 1 — Command-Line Fundamentals

Day	Lesson	Key commands
01	Introduction to Linux and PowerShell	`whoami` `hostname` `pwd` `ls` `uname` `date` · `Get-Location` `Get-ChildItem` `Get-ComputerInfo` `Get-Date`
02	File and Directory Management	`cd` `mkdir` `touch` `cp` `mv` `rm` · `Set-Location` `New-Item` `Copy-Item` `Move-Item` `Remove-Item`
03	Reading and Managing Files	`cat` `less` `head` `tail` `wc` · `Get-Content` `Select-Object` `Measure-Object` · redirection
04	Searching Files and Text	`find` `grep` `locate` `which` · `Get-ChildItem` `Select-String` `Where-Object` · regex
05	Command Help, History and Pipelines	`man` `history` `type` · `Get-Help` `Get-Command` `Get-History` · pipelines, exit codes

Module 2 — System Administration

Day	Lesson	Key commands
06	Users and Groups	`useradd` `usermod` `passwd` `id` `groups` · `Get-LocalUser` `New-LocalUser` `Add-LocalGroupMember`
07	File Permissions and Access Control	`chmod` `chown` `umask` · `Get-Acl` `Set-Acl` `icacls`
08	Processes and Services	`ps` `top` `htop` `kill` `systemctl` · `Get-Process` `Stop-Process` `Get-Service`
09	Disk and Storage Management	`df` `du` `lsblk` `mount` `findmnt` · `Get-Disk` `Get-Volume` `Get-PSDrive`
10	Software and Package Management	`apt` `dpkg` `dnf` · `winget` · PowerShellGet / PSResourceGet

Module 3 — Networking & Troubleshooting

Day	Lesson	Key commands
11	IP Configuration	`ip addr` `ip route` · `Get-NetIPConfiguration` `Get-NetIPAddress` `Get-NetRoute`
12	Connectivity Troubleshooting	`ping` `traceroute` `tracepath` `mtr` · `Test-Connection` `Test-NetConnection` `tracert`
13	DNS Troubleshooting	`dig` `nslookup` `host` · `Resolve-DnsName`
14	Ports, Sockets and Connections	`ss` `netstat` `lsof` · `Get-NetTCPConnection` `Get-NetUDPEndpoint`
15	Network Interfaces and Diagnostics	`ip link` `ip neigh` `tcpdump` · `Get-NetAdapter` `Get-NetNeighbor` `pktmon` · Wireshark

Module 4 — Security, Logs & Monitoring

Day	Lesson	Key commands
16	System Logs	`journalctl` `dmesg` `/var/log` · `Get-WinEvent` `wevtutil`
17	Authentication and Security Monitoring	`last` `lastb` `who` · auth.log · Security log event IDs 4624/4625/4672
18	Firewall Management	`ufw` `nft` `iptables` · `Get-NetFirewallProfile` `Get-NetFirewallRule`
19	System Security Auditing	`sudo -l` `find -perm` · `Get-ExecutionPolicy` `Get-MpComputerStatus`
20	Network Security Investigation	`nmap` `ss` `tcpdump` · `Get-NetTCPConnection` `pktmon`

Module 5 — Scripting & Automation

Day	Lesson	Key commands
21	Variables and User Input	`read` `echo` `export` · `Read-Host` `Write-Output` · scope, types
22	Conditions and Loops	`if` `case` `for` `while` · `if` `switch` `foreach` `ForEach-Object`
23	Functions and Reusable Scripts	Bash functions · advanced functions, `param()`, `try/catch`
24	Text and Data Processing	`awk` `sed` `cut` `sort` `uniq` · `Group-Object` `Sort-Object` · CSV and JSON
25	Task Scheduling and Automation	`cron` `crontab` systemd timers · Task Scheduler, `Register-ScheduledTask`

Module 6 — Professional Operations & Projects

Day	Lesson	Key commands
26	Remote Administration	`ssh` `ssh-keygen` `scp` `sftp` · `Enter-PSSession` `Invoke-Command` · WinRM and SSH remoting
27	Network Investigation	Case studies: TCP, routing, DNS, captures, Wireshark, Nmap
28	Advanced System Troubleshooting	CPU, memory, disk and network bottlenecks · root cause analysis
29	Integrated Diagnostic Automation	Full Linux and Windows diagnostic scripts · CSV/text reports
30	Final Capstone and Assessment	Complete investigation, final exam, certificate template

Suggested daily routine

1. Read the concepts (45–90 min).
2. Type every demonstration yourself rather than copy-pasting (60 min).
3. Complete the lab without looking at the solution, then compare (45–60 min).
4. Answer the practice questions and quiz (20 min).
5. Do the homework and tick the completion checklist.

Getting Started

This section prepares a safe practice environment for the next 30 days. Budget 1–2 hours.

Prerequisites

- Basic computer skills: installing software, using a web browser, editing a text file.
- A computer that can run **Windows 11** with **WSL2** or a virtual machine. 8 GB RAM minimum, 16 GB recommended. 40 GB free disk.
- An internet connection for installing packages.
- Administrator rights on **your own** machine, needed only for installation.

No prior command-line experience is required.

Recommended lab layout

Diagram — shown rendered on the website; text form:

```
flowchart TB
    subgraph Host["Your Windows 11 computer"]
        PS51["Windows PowerShell 5.1<br/>(built in)"]
        PS7["PowerShell 7.4+<br/>(install)"]
        WT["Windows Terminal"]
        subgraph WSL["WSL2"]
            U["Ubuntu 24.04 LTS<br/>Bash 5.2"]
        end
        subgraph VM["Optional VM (VirtualBox / Hyper-V)"]
            U2["Ubuntu 24.04 Server<br/>for networking & security labs"]
        end
    end
```

Option	Best for	Guide
Windows 11 + WSL2 Ubuntu (recommended)	Most learners. Both shells on one machine.	WSL2 Installation
Ubuntu 24.04 desktop or VM	Full Linux experience including systemd, firewall and networking labs	Ubuntu Setup
Windows 11 only	PowerShell-focused learners (use WSL2 for the Linux parts)	Windows Setup

WSL2 limitations

WSL2 is excellent for Days 1–7 and 21–24. Some later labs (firewall, systemd services, packet capture, network interfaces) work best in a **full Ubuntu VM**, and each lesson says when this applies.

Required software

Software	Required?	Purpose	Install guide
Windows Terminal	Recommended	Tabs for every shell	Built in to Windows 11
Windows PowerShell 5.1	Built in	Windows-only cmdlets, legacy compatibility	—
PowerShell 7.4+	Required	Modern, cross-platform PowerShell	PowerShell 7
WSL2 + Ubuntu 24.04	Required (or a VM)	Linux and Bash	WSL2
Visual Studio Code	Recommended	Editing scripts, with the PowerShell and ShellCheck extensions	Windows Setup
Git	Recommended	Clone this course, track your own scripts	Windows Setup
Wireshark, Nmap	From Day 15/20	Packet analysis, network discovery (lab only)	Installed in those lessons

Get the course files

```
# Windows (PowerShell) — requires Git
Set-Location ~
git clone https://github.com/<github-user>/linux-powershell-professional-mastery.git
```

```
# Ubuntu / WSL
cd ~
git clone https://github.com/<github-user>/linux-powershell-professional-mastery.git
```

Replace `<github-user>` with the account shown in the repository link at the top right of this site.

Before your first lab

Read **Lab Environment & Safety**. It sets out the rules every lab follows: authorised systems only, backups before configuration changes, and no disabling of security controls as a workaround.

Then start **Day 01**.

Windows 11 Setup

1. Check your Windows version

```
Get-ComputerInfo -Property OsName, OsVersion, OsBuildNumber
```

Any Windows 11 build (22000 or later) is supported. Windows 10 22H2 works for almost everything, but it is not the documented target.

2. Windows Terminal

Windows Terminal is included in Windows 11. Open it with Win → *Terminal*. Recommended settings (Ctrl+Comma):

- **Startup → Default profile:** *PowerShell* (PowerShell 7, after you install it).
- Keep separate profiles for *Windows PowerShell*, *PowerShell* and *Ubuntu*.

3. Confirm Windows PowerShell 5.1

```
$PSVersionTable.PSVersion
$PSVersionTable.PSEdition    # Desktop
```

4. Install PowerShell 7

Follow PowerShell 7 Installation. The short version is:

```
winget install --id Microsoft.PowerShell --source winget
```

Install failed with `Installer failed with exit code: 0x80072ee2`?

`0x80072EE2` is a **network timeout**. winget downloaded and verified the installer, but the MSI timed out while contacting Microsoft Update during setup. This is common behind proxies, firewalls, VPNs or on slow connections. Check your connection and retry. If it fails again, install without the Microsoft Update opt-in from an **elevated** terminal:

```
winget install --id Microsoft.PowerShell --source winget --override "/quiet /norestart
ADD_PATH=1 USE_MU=0 ENABLE_MU=0"
```

The full explanation, alternatives (manual MSI, Store, ZIP) and log collection steps are in Troubleshooting Windows installation.

5. Install Git and Visual Studio Code

```
winget install --id Git.Git --source winget
winget install --id Microsoft.VisualStudioCode --source winget
```

Close and reopen the terminal afterwards so the updated `PATH` is loaded, then verify:

```
git --version
code --version
```

Recommended VS Code extensions: **PowerShell** (`ms-vscode.powershell`), **ShellCheck** (`timonwong.shellcheck`), **WSL** (`ms-vscode-remote.remote-wsl`), **Markdown All in One**.

6. Script execution policy (read before changing anything)

On Windows client editions the default execution policy is **Restricted**, so `.ps1` scripts will not run. For this course set it **for your user only** to `RemoteSigned`. This allows your own local scripts and requires downloaded scripts to be signed or explicitly unblocked:

```
Get-ExecutionPolicy -List
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

> **Execution policy is not a security boundary**
>
> Execution policy prevents *accidental* script execution and is not designed to stop attackers. Never set `Unrestricted` or `Bypass` machine-wide. Day 19 covers this in depth.

7. Create your practice folder

```
New-Item -Path $HOME\lpm-lab -ItemType Directory -Force
```

Do all Windows practice work inside `$HOME\lpm-lab` unless a lab says otherwise.

Verification

```
$PSVersionTable.PSVersion          # 5.1.x in Windows PowerShell
pwsh -NoProfile -Command '$PSVersionTable.PSVersion'   # 7.x
git --version
Get-ExecutionPolicy -Scope CurrentUser                 # RemoteSigned
Test-Path $HOME\lpm-lab                                # True
```

Ubuntu 24.04 LTS Setup

Ubuntu 24.04 LTS ("Noble Numbat") is the documented Linux platform for this course. It receives standard security updates until 2029.

Option A — Virtual machine (recommended for Modules 3–4)

1. Download the ISO from https://ubuntu.com/download (Desktop or Server).
2. Create a VM in **VirtualBox** (free) or **Hyper-V** (Windows 11 Pro/Enterprise): 2 vCPU, 4 GB RAM, 25 GB disk.
3. Networking: use **NAT** for internet access. Add a second **Host-only** adapter for labs that need host-to-VM traffic (Days 12–15, 20, 26).
4. Install with default options and create a normal user (the course uses `student`).
5. **Take a snapshot** named `clean-install` once setup is finished. Before every risky lab (firewall, permissions), take another.

Option B — WSL2

See WSL2 Installation.

First-boot configuration

```
# Update the package lists and installed packages
sudo apt update && sudo apt full-upgrade -y

# Tools used throughout the course
sudo apt install -y curl wget git tree vim nano htop man-db bash-completion \
    net-tools dnsutils iproute2 iputils-ping traceroute mtr-tiny tcpdump \
    lsof psmisc sysstat jq shellcheck

# Check the release
cat /etc/os-release
```

What these packages provide

`dnsutils` provides `dig` and `nslookup` (Day 13). `iproute2` provides `ip` and `ss` (Days 11, 14). `mtr-tiny`, `traceroute` and `iputils-ping` are used for connectivity (Day 12), `tcpdump` for packet capture (Day 15), `sysstat` for `iostat`/`sar` (Day 28), `jq` for JSON (Day 24) and `shellcheck` for script linting (Module 5).

Install PowerShell 7 on Ubuntu (optional, from Day 5)

See PowerShell 7 Installation.

Create your practice folder

```
mkdir -p ~/lpm-lab
```

Verification

```
lsb_release -a          # Ubuntu 24.04.x LTS
bash --version | head -1
which dig ss ip tcpdump
ls -ld ~/lpm-lab
```

WSL2 Installation

The **Windows Subsystem for Linux 2** runs a real Linux kernel in a lightweight VM, integrated with Windows.

Requirements

- Windows 11 (any edition) or Windows 10 version 2004+.
- Virtualisation enabled in BIOS/UEFI. Check: Task Manager → Performance → CPU → *Virtualization: Enabled*.

Install

Open **Windows PowerShell as Administrator** and run:

```
wsl --install -d Ubuntu-24.04
```

Restart when prompted. After the restart Ubuntu finishes installing and asks you to create a **Linux user name and password**. These are separate from your Windows account.

Verify

```
wsl --list --verbose
```

Expected output (example):

```
  NAME            STATE           VERSION
* Ubuntu-24.04    Running         2
```

`VERSION` must be **2**. If it shows 1, run `wsl --set-version Ubuntu-24.04 2`.

Inside Ubuntu:

```
cat /etc/os-release | head -3
sudo apt update && sudo apt full-upgrade -y
```

Then install the course tools listed in Ubuntu Setup.

Useful WSL commands

Command (PowerShell)	Purpose
`wsl`	Start the default distribution
`wsl -d Ubuntu-24.04`	Start a specific distribution
`wsl --shutdown`	Stop all distributions (fixes many networking glitches)

Command (PowerShell)	Purpose
`wsl --update`	Update the WSL kernel and components
`wsl --export Ubuntu-24.04 backup.tar`	Back up the whole distribution
`wsl --status`	Show default version and settings

Files across both systems

- Windows drives appear in Linux under `/mnt/c`, `/mnt/d`.
- Linux files appear in Windows Explorer at `\\wsl$\Ubuntu-24.04\home\<user>`.
- **Keep Linux lab work in the Linux home directory** (`~`), not under `/mnt/c`. Performance is much better, and Linux permissions (Day 7) behave correctly.

Known limitations for this course

Topic	WSL2 behaviour	Recommendation
systemd services (Day 8)	Supported. Ubuntu 24.04 on WSL enables systemd by default	OK
UFW / nftables firewall (Day 18)	Works inside WSL, but does not protect Windows	Use a VM for realism
Packet capture (Days 15, 20, 27)	`tcpdump` sees only the WSL virtual NIC	Use a VM for full captures
Network interfaces (Day 15)	Single virtual adapter (NAT or mirrored mode)	VM for multi-NIC labs
Login history `last`/`lastb` (Day 17)	Limited because WSL logins are not real logins	VM recommended

PowerShell 7 Installation

PowerShell 7 installs **side by side** with Windows PowerShell 5.1. It does not replace it. The executable is `pwsh`. Any supported 7.x release works for this course. `winget` installs the latest stable release (7.6 at the time of writing).

Windows 11

winget (recommended)

```
winget search --id Microsoft.PowerShell
winget install --id Microsoft.PowerShell --source winget
```

MSI package

Download the latest `PowerShell-7.x.x-win-x64.msi` from https://github.com/PowerShell/PowerShell/releases and run it. Keep the defaults, and enable *Add PowerShell to PATH*.

Microsoft Store

Search for **PowerShell** by Microsoft Corporation. It updates automatically. Note that some system-level tasks behave differently in the Store version.

Verify in a **new** terminal:

```
pwsh -NoProfile -Command '$PSVersionTable'
```

Troubleshooting Windows installation

`Installer failed with exit code: 0x80072ee2`

Symptom (reported by a learner on Windows 11):

```
PS ~ > winget install --id Microsoft.PowerShell --source winget
Found PowerShell [Microsoft.PowerShell] Version 7.6.9.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Successfully verified installer hash
Starting package install...
Installer failed with exit code: 0x80072ee2 : unknown error
```

What it means: `0x80072EE2` is the Windows networking error `ERROR_INTERNET_TIMEOUT` ("The operation timed out"). The **download and hash check already succeeded**, so the failure happened while the MSI installer itself was running. By default, the PowerShell MSI contacts the network during setup to opt you in to updates through **Microsoft Update** (the `USE_MU` / `ENABLE_MU` installer options). On a slow

connection, behind a proxy or firewall, or on a VPN, that request can time out and the installation rolls back.

Fix it, in this order:

1. **Check the connection, then retry once.**

   ```
   Test-NetConnection -ComputerName www.microsoft.com -Port 443   # TcpTestSucceeded should be True
   netsh winhttp show proxy                                       # system (WinHTTP) proxy settings
   ```

   Disconnect from VPNs and metered or captive-portal Wi-Fi, then run the `winget install` command again.

2. **Install without the Microsoft Update opt-in.** This avoids the network call that timed out. Run it in an **elevated** terminal:

   ```
   winget install --id Microsoft.PowerShell --source winget --override
   "/quiet /norestart ADD_PATH=1 USE_MU=0 ENABLE_MU=0"
   ```

   `--override` **replaces** winget's default installer arguments, which is why `/quiet` and `ADD_PATH=1` are included explicitly. You then keep PowerShell updated with `winget upgrade --id Microsoft.PowerShell` instead of Microsoft Update.

3. **Use the MSI directly.** Download `PowerShell-7.x.x-win-x64.msi` from https://github.com/PowerShell/PowerShell/releases, then either run it and **untick** the two *Microsoft Update* checkboxes, or install silently from an elevated terminal:

   ```
   msiexec.exe /package .\PowerShell-7.6.9-win-x64.msi /quiet ADD_PATH=1 USE_MU=0 ENABLE_MU=0
   ```

   Adjust the file name to the version you downloaded.

4. **Use an alternative package.** Install from the Microsoft Store (`winget install --id 9MZ1SNWT0N5D --source msstore`), or use the portable **ZIP** package from the same releases page if you cannot install software system-wide.

Collect diagnostics if it still fails:

```
winget install --id Microsoft.PowerShell --source winget --log "$env:TEMP\pwsh-install.log"
winget --logs          # opens winget's log folder
```

Search the log for `Return value 3` (the MSI step that failed) and for the URL that timed out.

Verify success:

```
winget list --id Microsoft.PowerShell
pwsh -NoProfile -Command '$PSVersionTable.PSVersion'
```

> **Validation status**
>
> This fix is based on the documented PowerShell MSI options (`ADD_PATH`, `USE_MU`, `ENABLE_MU`) and the Windows error code definition. It has not been reproduced in CI, because the failure depends on the local network.

Other common Windows installation problems

Symptom	Cause	Fix
`'winget' is not recognized`	App Installer missing or outdated	Update **App Installer** from the Microsoft Store, then open a new terminal
`pwsh` not recognized after a successful install	`PATH` not refreshed in the current window	Close and reopen the terminal, or run `& "$env:ProgramFiles\PowerShell\7\pwsh.exe"`
`0x80070005` / access denied	Installing system-wide without elevation	Run the terminal **as Administrator** for the install only
Install blocked by company policy	Software restriction or Intune/ AppLocker rules	Ask IT, or use the Store/ZIP package if policy allows

Ubuntu 24.04

Install from Microsoft's package repository:

```
# Prerequisites
sudo apt update && sudo apt install -y wget apt-transport-https software-properties-common

# Register the Microsoft repository (reads VERSION_ID from /etc/os-release)
source /etc/os-release
wget -q "https://packages.microsoft.com/config/ubuntu/$VERSION_ID/packages-microsoft-prod.deb"
sudo dpkg -i packages-microsoft-prod.deb
rm packages-microsoft-prod.deb

# Install
sudo apt update && sudo apt install -y powershell

# Start
pwsh
```

Alternatively: `sudo snap install powershell --classic`.

Which edition should I use?

Situation	Use
Learning, new scripts, cross-platform work	**PowerShell 7**
Modules that only run on .NET Framework (some older RSAT/Exchange tooling)	Windows PowerShell 5.1
Group Policy logon scripts on older fleets	Windows PowerShell 5.1 (always present)

Every lesson marks commands that are **5.1-only**, **7-only** or **Windows-only**.

Keep it updated

```
winget upgrade --id Microsoft.PowerShell
```

```
sudo apt update && sudo apt install --only-upgrade powershell
```

Official documentation: https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell

Lab Environment & Safety Rules

Every lab in this course follows these rules. They reflect how professional IT and security teams work.

1. Authorised systems only

- Run labs on **localhost**, **WSL2**, **your own virtual machines**, or systems you have **explicit written permission** to test.
- **Never** scan, probe, brute-force or capture traffic on networks or hosts you do not own or administer. That includes employer, school, public Wi-Fi and internet hosts. Unauthorised scanning is illegal in many countries (for example under the IT Act 2000 in India and the Computer Misuse Act in the UK).
- Network discovery labs (Nmap, Days 20 and 27) target `127.0.0.1`, your own VMs, or `scanme.nmap.org`, which the Nmap project explicitly permits for light testing.

2. Snapshots and backups before changes

Before any lab that changes **permissions, firewall rules, services, users or system configuration**:

- **VM:** take a snapshot.
- **WSL:** `wsl --export Ubuntu-24.04 $HOME\wsl-backup.tar`
- **Windows:** create a restore point (`Checkpoint-Computer -Description "Before Day 18 lab"` in an elevated Windows PowerShell 5.1), and export what you will change (for example `netsh advfirewall export C:\Backup\fw.wfw`).
- **Files:** copy the original with a timestamp (`cp -a file file.bak.$(date +%F)`).

3. No real credentials

- Use throwaway lab accounts (`student`, `labuser1`) and passwords that you use nowhere else.
- **Never** paste real passwords, API keys, tokens or private keys into scripts, screenshots, tickets or this course's repository.

4. Destructive commands are always explained first

Commands that delete, overwrite or disrupt (`rm -r`, `Remove-Item -Recurse`, `kill -9`, `Stop-Service`, firewall changes) are marked with a warning, explained, and preceded by a **preview** (`ls`, `-WhatIf`) step.

5. Never disable security controls as a workaround

If a lab fails because of the firewall, antivirus, SELinux/AppArmor or execution policy, **diagnose why**. Do not turn the control off. Where a control must be adjusted, the lesson shows the narrowest, reversible change, and how to undo it.

6. Least privilege

Work as a normal user. Use `sudo` or an elevated PowerShell only for the specific command that needs it, then return to the normal session.

Recommended folder layout

```
~/lpm-lab/            # Linux practice root
$HOME\lpm-lab\        # Windows practice root
    day01/ day02/ ...  # one folder per day
    reports/           # output of course scripts
```

Day 01: Introduction to Linux and PowerShell

Lesson overview

Module	1 — Command-Line Fundamentals
Difficulty	Beginner
Estimated time	3–4 hours (90 min reading, 60 min demonstrations, 60–90 min lab and quiz)
Platforms	Ubuntu 24.04 LTS (Bash 5.2) · Windows 11 (Windows PowerShell 5.1 and PowerShell 7.4+)
Lab	Lab 01: First Contact

Learning Objectives

By the end of this lesson you will be able to:

1. Explain the difference between a **GUI** and a **CLI**, and between a **terminal**, a **shell** and the **operating system kernel**.
2. Describe what **Bash** and **PowerShell** are, and the key design difference between them (text streams vs .NET objects).
3. Open a terminal on Ubuntu and a PowerShell session on Windows, and tell which PowerShell edition you are running.
4. Run your first identity and system commands: `whoami`, `hostname`, `pwd`, `ls`, `uname`, `date` and their PowerShell counterparts `Get-Location`, `Get-ChildItem`, `Get-ComputerInfo`, `Get-Date`.
5. Read command syntax: command name, options/parameters, arguments.
6. Use built-in help (`man`, `--help`, `Get-Help`, `Get-Command`) to learn any command without searching the web.

Prerequisites

- A computer with **Ubuntu 24.04 LTS** (physical, VM or WSL2) and/or **Windows 11**.
- Optional but recommended: PowerShell 7 installed alongside Windows PowerShell 5.1.
- No prior command-line experience is needed.
- Read Lab Environment & Safety once before doing any lab in this course.

Introduction and Concepts

GUI versus CLI

A **graphical user interface (GUI)** lets you operate a computer by pointing and clicking: File Explorer, Settings, the Ubuntu Files app. A **command-line interface (CLI)** lets you operate it by typing instructions as text.

The GUI is easier to discover; the CLI is far more powerful for professional work because commands are:

Property	Why it matters to an IT professional
Precise	`ls -lh /var/log` always means exactly the same thing. "Click the third icon" does not.
Repeatable	You can run the same command on 1 or 1,000 servers.
Scriptable	Commands saved in a file become automation (Module 5).
Remote-friendly	Most servers have no GUI at all. You manage them over SSH or PowerShell Remoting (Day 26).
Auditable	Commands can be logged, reviewed and pasted into a ticket or incident report.
Lightweight	A text session works over a slow VPN where remote desktop would freeze.

Terminal, shell and operating system

These three words are often mixed up. They are different layers:

Diagram — shown rendered on the website; text form:

```
flowchart LR
    U[You type a command] --> T["Terminal<br/>(window that shows text)<br/>e.g. GNOME Terminal,
Windows Terminal"]
    T --> S["Shell<br/>(interprets the command)<br/>e.g. Bash, PowerShell"]
    S --> K["Operating system kernel<br/>(does the real work)<br/>Linux kernel / Windows NT kernel"]
    K --> H[Hardware: CPU, disk, network]
    K -->|results| S
    S -->|text output| T
```

- **Terminal (terminal emulator):** the window program. It draws characters, handles copy/paste, tabs and fonts. Examples: GNOME Terminal, Konsole, Windows Terminal, the VS Code terminal. It does **not** understand commands.
- **Shell:** the program running *inside* the terminal. It reads what you type, expands wildcards and variables, finds the program to run, starts it and shows the result. Examples: Bash, Zsh, PowerShell, cmd.exe.
- **Operating system (kernel):** manages processes, memory, files, devices and the network. The shell asks the kernel to do things through *system calls*.

You can run different shells in the same terminal: Windows Terminal can host PowerShell 7, Windows PowerShell 5.1, cmd.exe and an Ubuntu (WSL) Bash tab side by side.

What is Bash?

Bash (Bourne-Again SHell) is the default interactive shell on Ubuntu and most Linux distributions. It is a GNU project, first released in 1989.

Key characteristics:

- Commands are mostly **separate programs** (`ls` is `/usr/bin/ls`). Bash starts them and connects their input and output.
- Data flows between commands as **plain text** (bytes). You extract fields with tools such as `grep`, `cut` and `awk` (Days 4 and 24).

- Commands are short and terse (`ls`, `cp`, `rm`) and options are usually single letters (`-l`) or long words (`--all`).
- Bash is **case-sensitive**: `ls` works, `LS` does not. File names are case-sensitive too: `Report.txt` and `report.txt` are two different files.

What is PowerShell?

PowerShell is Microsoft's task-automation shell and scripting language, built on .NET. There are two editions you will meet:

Edition	Executable	Version	Runtime	Platforms	Status
Windows PowerShell	`powershell.exe`	5.1	.NET Framework 4.x	Windows only (built in)	Maintained, no new features
PowerShell (often "PowerShell 7")	`pwsh` / `pwsh.exe`	7.x	.NET (Core)	Windows, Linux, macOS	Actively developed

Key characteristics:

- Commands are called **cmdlets** and follow a **Verb-Noun** naming pattern: `Get-Process`, `Stop-Service`, `New-Item`. Run `Get-Verb` to see the approved verbs.
- Cmdlets output **.NET objects**, not text. An object has **properties** (e.g. a file's `Name`, `Length`, `LastWriteTime`) that you select, filter and sort directly without parsing text (Day 5).
- Cmdlet and parameter names are **case-insensitive**: `get-childitem`, `Get-ChildItem` and `GET-CHILDITEM` are identical.
- Parameters start with a dash and are whole words: `-Path`, `-Recurse`. You may shorten them to any unambiguous prefix (`-Rec`), and tab completion fills them in.
- **Aliases** make it feel familiar: `ls`, `dir`, `cd`, `pwd`, `cat`, `cp`, `mv`, `rm` are aliases on **Windows**. Run `Get-Alias ls` to see what an alias points to.

> **Aliases differ between Windows and Linux PowerShell**
>
> In PowerShell 7 **on Linux and macOS**, aliases such as `ls`, `cp`, `mv`, `rm`, `cat` and `ps` are **not** defined, so they run the native Linux programs instead. Always use full cmdlet names (`Get-ChildItem`) in scripts so they behave the same everywhere.

Opening a terminal

Ubuntu 24.04

- **Keyboard shortcut:** Ctrl+Alt+T opens GNOME Terminal.
- **Activities menu:** press the Super key, type `terminal`, press Enter.
- **WSL2 on Windows:** open Windows Terminal and choose the *Ubuntu* tab, or type `wsl` in PowerShell.
- **Remote server:** `ssh user@server` (covered fully on Day 26).

The prompt looks like this:

```
student@ubuntu-lab:~$
```

`student` is the user, `ubuntu-lab` is the host name, `~` is the current directory (your home) and `$` means a normal user. A `#` instead of `$` means you are **root** (the administrator), so be careful.

Windows 11

- **Start menu:** press Win, type `terminal` (Windows Terminal) or `powershell`, press Enter.
- **As administrator:** right-click *Terminal* → *Run as administrator*, or press Win+X then A. Use this only when a task really needs elevation.
- **PowerShell 7:** run `pwsh` or pick *PowerShell* (not *Windows PowerShell*) in Windows Terminal.
- **Run dialog:** Win+R, type `powershell` or `pwsh`.

The prompt looks like this:

```
PS C:\Users\student>
```

`PS` means you are in PowerShell, followed by the current location. An elevated window shows *Administrator:* in its title bar.

Command syntax

Almost every command in both shells follows this shape:

```
command  [options / parameters]  [arguments]
```

Bash

```
ls -l -a /etc          # command=ls, options=-l -a, argument=/etc
ls -la /etc            # short options can be combined
ls --all --human-readable -l /var/log   # long options use two dashes
```

PowerShell

```
Get-ChildItem -Path C:\Windows -Directory     # -Path takes a value; -Directory is a switch
Get-ChildItem C:\Windows -Directory           # -Path is positional, so the name can be omitted
Get-ChildItem -Path $HOME -Filter *.txt -Recurse
```

PowerShell parameters are one of two kinds. A **parameter with a value** looks like `-Path C:\Windows`. A **switch** is either present or absent, like `-Recurse` or `-Force`.

In help pages and in this course, `[square brackets]` mean *optional*, `<angle brackets>` mean *replace with your value*, and `...` means *may be repeated*.

Linux Command Reference

Command	Purpose	Package / availability	Needs root?
`whoami`	Print the effective user name	coreutils (always installed)	No
`hostname`	Show or set the system host name	hostname (installed by default)	Only to set
`hostnamectl`	Show or set host name and OS details (systemd)	systemd	Only to set
`pwd`	Print the current working directory	Bash built-in and coreutils	No
`ls`	List directory contents	coreutils	No (only to read protected dirs)
`uname`	Print kernel and system information	coreutils	No
`date`	Print or set the date and time	coreutils	Only to set
`man`	Display the manual page for a command	man-db	No
`<command> --help`	Short built-in usage summary	Most GNU programs	No
`help`	Help for Bash *built-in* commands (`cd`, `pwd`, `type`)	Bash built-in	No
`type` / `which`	Show what a command name resolves to	Bash built-in / debianutils	No

PowerShell Command Reference

Cmdlet	Common aliases	Purpose	Works in 5.1	Works in 7.x	Admin?
`Get-Location`	`pwd`, `gl`	Show the current location	Yes	Yes (all OS)	No
`Get-ChildItem`	`ls*`, `dir`, `gci`	List items in a location	Yes	Yes (all OS)	No
`Get-ComputerInfo`	`gin`	Detailed OS and hardware information	Yes	**Windows only**	No
`Get-Date`	—	Current date/time as a `DateTime` object	Yes	Yes (all OS)	No
`Get-Help`	`help`, `man*`	Show help for cmdlets and concepts	Yes	Yes	`Update-Help` may need admin in 5.1
`Get-Command`	`gcm`	Find commands by name, verb, noun or module	Yes	Yes	No
`$env:USERNAME`, `whoami.exe`	—	Current user	Yes	Yes (`$env:USER` on Linux)	No

Cmdlet	Common aliases	Purpose	Works in 5.1	Works in 7.x	Admin?
`$env:COMPUTERNAME`, `hostname.exe`	—	Computer name	Yes	`hostname` on Linux	No
`$PSVersionTable`	—	PowerShell version and edition	Yes	Yes	No

* `ls` and `man` are aliases only on Windows. On Linux they run the native programs.

Detailed Command Explanations

`whoami` — who am I logged in as?

`whoami` prints the **effective** user name, which is the account whose permissions your commands are running with right now. That is not always the person who logged in: after `sudo -i` or `su - admin` it changes.

On Windows, `whoami.exe` exists too and prints `COMPUTER\user` or `DOMAIN\user`. In PowerShell the environment variable `$env:USERNAME` holds the plain name, and `[Security.Principal.WindowsIdentity]::GetCurrent().Name` returns the full identity.

Why professionals use it: before running any administrative change, confirm *who* you are. Running something as root by mistake, or failing because you are not, are both everyday mistakes.

`hostname` — which machine am I on?

`hostname` prints the system's host name. When you have five SSH sessions open, it is the fastest way to confirm you are on `web-prod-02` and **not** `db-prod-01` before you restart something. `hostname -I` (capital i, Linux only) prints all IP addresses assigned to the host. On systemd systems `hostnamectl` shows the host name together with the OS, kernel and hardware vendor.

On Windows, `hostname.exe` works in both PowerShell and cmd. PowerShell also exposes `$env:COMPUTERNAME`.

`pwd` / `Get-Location` — where am I?

`pwd` (print working directory) shows the absolute path of the directory you are "in". Every relative path you type is interpreted relative to this location (Day 2). `pwd -P` resolves symbolic links and shows the physical path.

`Get-Location` returns a `PathInfo` object. Its `Path` property is the location. PowerShell locations can be on *providers* other than the file system, such as the registry (`HKLM:\`) or certificate store (`Cert:\`), which is why the cmdlet is not called `Get-Directory`.

`ls` / `Get-ChildItem` — what is here?

`ls` lists the contents of a directory (the current one if you give no argument). Without options it shows names only. Most useful options:

- `-l` long format: permissions, link count, owner, group, size, modification time, name
- `-a` include hidden entries (names starting with `.`)
- `-h` human-readable sizes (K, M, G), used with `-l`

- `-t` sort by modification time, newest first; add `-r` to reverse
- `-R` recurse into subdirectories
- `-d` list a directory itself, not its contents

`Get-ChildItem` returns **objects**: `FileInfo` for files and `DirectoryInfo` for directories. The table you see is just PowerShell's default *formatting* of those objects. The objects themselves carry many more properties (`FullName`, `Extension`, `CreationTime`, `Attributes`...).

`uname` — what kernel and architecture?

`uname` prints information about the running kernel. `uname -a` prints everything; `-s` kernel name, `-n` host name, `-r` kernel release, `-m` machine architecture (`x86_64`, `aarch64`), `-o` operating system. To see the **distribution** (Ubuntu version), read `/etc/os-release` or run `lsb_release -a`, because `uname` describes the kernel only.

The nearest PowerShell equivalents are `Get-ComputerInfo` (Windows) and the automatic variables `$IsLinux`, `$IsWindows`, `$IsMacOS` (PowerShell 7 only).

`Get-ComputerInfo` — full system inventory (Windows)

`Get-ComputerInfo` (PowerShell 5.1+, Windows only) collects about 180 properties: OS name and build, install date, BIOS, manufacturer, model, CPU count, memory, time zone, domain and more. It queries WMI/CIM, so it takes several seconds. Always ask for just the properties you need with `-Property`. Property names accept wildcards (`-Property Os*`).

On PowerShell 7 for Linux this cmdlet does not exist. Use `uname`, `hostnamectl` and `/etc/os-release` instead.

`date` / `Get-Date` — what time is it, exactly?

Time matters enormously in IT operations: log correlation, certificate validity, Kerberos authentication (which fails when clocks are skewed by more than 5 minutes by default) and incident timelines.

- `date` prints the current date and time in the local time zone. `date -u` prints UTC. `date +FORMAT` formats it, e.g. `date '+%Y-%m-%d %H:%M:%S'`. GNU `date -d 'yesterday'` or `date -d '2 hours ago'` calculates other times. `date -Iseconds` prints ISO 8601. `timedatectl` shows time zone and NTP synchronisation status.
- `Get-Date` returns a `DateTime` **object** that you can do arithmetic on: `(Get-Date).AddDays(-7)`. Format it with `-Format 'yyyy-MM-dd HH:mm:ss'` (.NET format strings) or `-UFormat '%Y-%m-%d'` (Unix-style). UTC: `(Get-Date).ToUniversalTime()`, or `Get-Date -AsUTC` in PowerShell 7.1+.

> **Use ISO 8601 and UTC in reports**
>
> `2026-11-08T10:05:00Z` sorts correctly as text and means the same thing in every country. Day 29's report scripts use this format.

Getting help — the most important skill of the day

Bash / Linux

- `man ls` opens the manual page. Navigate with arrow keys or Space, search with `/word` then N for the next match, quit with Q. Sections: 1 = user commands, 5 = file formats, 8 = admin commands (`man 5 passwd` vs `man 1 passwd`).
- `ls --help` prints a short usage summary directly to the terminal.
- `help cd` documents Bash **built-ins**, which have no separate manual page.
- `type ls` tells you whether a name is an alias, built-in, function or file. `which ls` prints the path of the executable.
- `apropos network` / `man -k network` searches manual page descriptions.
- `whatis uname` prints a one-line description.

PowerShell

- `Get-Help Get-ChildItem` shows a summary. Add `-Examples`, `-Detailed`, `-Full` or `-Parameter Path`.
- `Get-Help Get-ChildItem -Online` opens the official documentation in a browser.
- `Update-Help` downloads the full local help files. On Windows PowerShell 5.1 run it from an **elevated** window. On PowerShell 7 use `Update-Help -Scope CurrentUser` without admin rights. Until you do this, `Get-Help` shows only auto-generated syntax.
- `Get-Command *service*` finds commands by wildcard. `Get-Command -Verb Get -Noun Net*` searches by verb/noun.
- `Get-Help about_*` lists conceptual topics, e.g. `Get-Help about_Wildcards`.

Syntax and Parameters

Bash commands

```
whoami
hostname [-I] [-f] [-s]
pwd [-L | -P]
ls [OPTION]... [FILE]...
uname [-a] [-s] [-n] [-r] [-m] [-o]
date [-u] [-d STRING] [+FORMAT]
man [SECTION] PAGE
```

Option	Command	Meaning
`-I`	hostname	All IP addresses of the host (Linux `hostname` from the *hostname* package)
`-f` / `-s`	hostname	Fully-qualified / short name
`-P`	pwd	Physical path with symlinks resolved
`-l`, `-a`, `-h`, `-t`, `-r`, `-R`, `-d`	ls	Long, all, human sizes, sort by time, reverse, recursive, directory itself
`-a`, `-s`, `-n`, `-r`, `-m`, `-o`	uname	All, kernel name, node name, kernel release, machine, OS

Option	Command	Meaning
`-u`	date	Use UTC
`-d STRING`	date	Display a described time (`'tomorrow'`, `'2026-01-01 09:00'`) — GNU only
`+FORMAT`	date	`%Y` year, `%m` month, `%d` day, `%H:%M:%S` time, `%Z` zone, `%s` Unix epoch

PowerShell cmdlets

```
Get-Location [-PSProvider <String[]>] [-PSDrive <String[]>] [-Stack]
Get-ChildItem [[-Path] <String[]>] [[-Filter] <String>] [-Include <String[]>] [-Exclude <String[]>]
              [-Recurse] [-Depth <UInt32>] [-Force] [-Name] [-File] [-Directory] [-Hidden]
Get-ComputerInfo [[-Property] <String[]>]
Get-Date [[-Date] <DateTime>] [-Format <String>] [-UFormat <String>] [-AsUTC]
Get-Help [[-Name] <String>] [-Examples | -Detailed | -Full | -Online | -Parameter <String>]
Get-Command [[-Name] <String[]>] [-Verb <String[]>] [-Noun <String[]>] [-Module <String[]>]
```

Parameter	Cmdlet	Meaning
`-Path`	Get-ChildItem	Location(s) to list; supports wildcards. Positional (position 0).
`-Filter`	Get-ChildItem	Provider-side filter such as `*.log`. It is faster than `-Include` because the file system applies it.
`-Recurse` / `-Depth`	Get-ChildItem	Recurse into subfolders / limit the recursion depth (PowerShell 5.0+)
`-Force`	Get-ChildItem	Include hidden and system items (similar to `ls -a`)
`-File` / `-Directory`	Get-ChildItem	Only files / only folders
`-Name`	Get-ChildItem	Return names as plain strings instead of objects
`-Property`	Get-ComputerInfo	Only collect these properties (wildcards allowed)
`-Format`	Get-Date	.NET format string: `yyyy-MM-dd HH:mm:ss`, `o` (round-trip ISO 8601). The output becomes a **string**, not a DateTime.
`-UFormat`	Get-Date	Unix-style format: `%Y-%m-%d`
`-AsUTC`	Get-Date	Return UTC time (PowerShell 7.1+ only)

Practical Demonstrations

The outputs below are **examples**. Your user name, host name, dates and sizes will be different. PowerShell outputs were captured on Windows 11 (build 26200) with Windows PowerShell 5.1, with the user and host renamed to `student` and `WIN11-LAB`. Linux outputs show the standard format on Ubuntu 24.04 LTS, with the user `student` and host `ubuntu-lab`.

Demo 1 — Identify yourself and the machine

Bash

```
whoami
hostname
hostname -I
```

Expected output (example):

```
student
ubuntu-lab
192.168.56.10 10.0.2.15
```

PowerShell

```
whoami
$env:USERNAME
hostname
$env:COMPUTERNAME
```

Expected output (example):

```
win11-lab\student
student
WIN11-LAB
WIN11-LAB
```

Demo 2 — Where am I and what is here?

Bash

```
pwd
ls
ls -la
ls -lh /var/log | head -5
```

Expected output (example):

```
/home/student
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
total 48
drwxr-x--- 15 student student 4096 Nov  8 09:12 .
drwxr-xr-x  3 root    root    4096 Oct 30 14:01 ..
-rw-------  1 student student 1342 Nov  8 09:10 .bash_history
-rw-r--r--  1 student student  220 Oct 30 14:01 .bash_logout
-rw-r--r--  1 student student 3771 Oct 30 14:01 .bashrc
drwxr-xr-x  2 student student 4096 Oct 30 14:20 Desktop
...
total 2.9M
-rw-r--r--  1 root      root       11K Nov  8 08:59 alternatives.log
drwxr-xr-x  2 root      root      4.0K Nov  3 06:25 apt
-rw-r-----  1 syslog    adm        38K Nov  8 09:17 auth.log
-rw-r--r--  1 root      root       59K Oct 30 14:05 bootstrap.log
```

How to read a long-listing line: `-rw-r--r--` is the type and permissions (Day 7), `1` the link count, `root root` the owner and group, `11K` the size, `Nov 8 08:59` the modification time, then the name.

PowerShell

```
Get-Location
Get-ChildItem C:\Windows -Directory | Select-Object -First 3
Get-ChildItem $HOME -Force
```

Expected output (captured, Windows PowerShell 5.1):

```
Path
----
C:\Users\student

    Directory: C:\Windows

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11-09-2026     14:11                appcompat
d-----        06-11-2026     18:48                apppatch
d-----        08-11-2026     04:41                AppReadiness
```

The `Mode` column letters are: `d` directory, `a` archive, `r` read-only, `h` hidden, `s` system, `l` reparse point/link. The date format follows your Windows regional settings. The capture above used an Indian locale (dd-MM-yyyy).

Demo 3 — Kernel, OS and hardware

Bash

```
uname -a
uname -r
uname -m
cat /etc/os-release
hostnamectl
```

Expected output (example):

```
Linux ubuntu-lab 6.8.0-48-generic #48-Ubuntu SMP PREEMPT_DYNAMIC Fri Sep 27 14:04:52 UTC 2024
x86_64 x86_64 x86_64 GNU/Linux
6.8.0-48-generic
x86_64
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
...
 Static hostname: ubuntu-lab
       Icon name: computer-vm
         Chassis: vm
  Virtualization: oracle
Operating System: Ubuntu 24.04.1 LTS
          Kernel: Linux 6.8.0-48-generic
    Architecture: x86-64
```

PowerShell

```powershell
$PSVersionTable
Get-ComputerInfo -Property OsName, OsVersion, OsArchitecture, CsNumberOfLogicalProcessors
```

Expected output (captured, Windows PowerShell 5.1):

```
Name                           Value
----                           -----
PSVersion                      5.1.26100.9168
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.26100.9168
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1

OsName                                     OsVersion  OsArchitecture CsNumberOfLogicalProcessors
------                                     ---------  -------------- ---------------------------
Microsoft Windows 11 Home Single Language 10.0.26200 64-bit                                    8
```

`PSEdition` is `Desktop` for Windows PowerShell 5.1 and `Core` for PowerShell 7.

Demo 4 — Dates and times

Bash

```bash
date
date -u
date '+%Y-%m-%d %H:%M:%S'
date -Iseconds
date -d '7 days ago' +%F
timedatectl
```

Expected output (example, time zone Asia/Kolkata):

```
Sun Nov  8 15:35:43 IST 2026
Sun Nov  8 10:05:43 UTC 2026
2026-11-08 15:35:43
2026-11-08T15:35:43+05:30
2026-11-01
               Local time: Sun 2026-11-08 15:35:43 IST
           Universal time: Sun 2026-11-08 10:05:43 UTC
                 RTC time: Sun 2026-11-08 10:05:43
                Time zone: Asia/Kolkata (IST, +0530)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

PowerShell

```powershell
Get-Date
Get-Date -Format 'yyyy-MM-dd HH:mm:ss'
Get-Date -Format o
(Get-Date).ToUniversalTime()
```

```
(Get-Date).AddDays(-7).ToString('yyyy-MM-dd')
(Get-Date).DayOfWeek
```

Expected output (example):

```
08 November 2026 15:35:43
2026-11-08 15:35:43
2026-11-08T15:35:43.1234567+05:30
08 November 2026 10:05:43
2026-11-01
Sunday
```

Demo 5 — Learning a command you have never seen

Bash

```
type ls pwd cd
whatis uname
uname --help | head -15
man -k "host name"
```

Expected output (example):

```
ls is aliased to `ls --color=auto'
pwd is a shell builtin
cd is a shell builtin
uname (1)            - print system information
Usage: uname [OPTION]...
Print certain system information.  With no OPTION, same as -s.

  -a, --all                print all information, in the following order,
                             except omit -p and -i if unknown:
  -s, --kernel-name        print the kernel name
  -n, --nodename           print the network node hostname
  -r, --kernel-release     print the kernel release
...
hostname (1)         - show or set the system's host name
hostnamectl (1)      - Control the system hostname
```

PowerShell

```
Get-Command Get-ChildItem
Get-Alias ls, dir, cd, pwd, cat
Get-Command -Verb Get -Noun *Computer*
Get-Help Get-Date -Examples
```

Expected output (captured, Windows PowerShell 5.1):

```
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          Get-ChildItem                                      3.1.0.0
Microsoft.PowerShell.Management

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           ls -> Get-ChildItem
Alias           dir -> Get-ChildItem
Alias           cd -> Set-Location
Alias           pwd -> Get-Location
```

```
Alias           cat -> Get-Content
...
```

Linux vs PowerShell Comparison

Task	Bash (Ubuntu)	PowerShell (Windows)	Notes
Current user	`whoami`	`whoami` · `$env:USERNAME`	`whoami.exe` prints `HOST\user`
Host name	`hostname`	`hostname` · `$env:COMPUTERNAME`	
Current directory	`pwd`	`Get-Location` (`pwd`)	PS returns an object; use `(Get-Location).Path` for the string
List a directory	`ls -la`	`Get-ChildItem -Force` (`ls`, `dir`)	PS objects have more properties than the table shows
Kernel / OS info	`uname -a`, `cat /etc/os-release`	`Get-ComputerInfo -Property Os*`	No single exact equivalent; different OS models
Architecture	`uname -m`	`$env:PROCESSOR_ARCHITECTURE`	Windows reports `AMD64` / `ARM64`
Shell version	`echo $BASH_VERSION`	`$PSVersionTable.PSVersion`	
Date and time	`date`	`Get-Date`	PS returns a `DateTime` you can calculate with
Formatted date	`date +%F`	`Get-Date -Format yyyy-MM-dd`	Different format languages
UTC time	`date -u`	`(Get-Date).ToUniversalTime()` / `Get-Date -AsUTC` (7.1+)	
Full help	`man ls`	`Get-Help Get-ChildItem -Full`	Run `Update-Help` once in PowerShell
Quick help	`ls --help`	`Get-Help Get-ChildItem` · `Get-ChildItem -?`	
Find a command	`apropos disk`, `compgen -c | grep disk`	`Get-Command *disk*`	
What is this name?	`type ls`	`Get-Command ls`	
Case sensitivity	Commands and file names are case-sensitive	Cmdlets case-insensitive; NTFS names case-insensitive by default	
Data between commands	Text	.NET objects	The central difference, explored on Day 5

Real-World IT Scenarios

1. **The "wrong server" incident.** An engineer intends to reboot a test server but runs the command in the production SSH tab. Professionals run `hostname` (and often put the host name in their prompt) before every disruptive action. Many companies colour production prompts red for this reason.

2. **Help-desk ticket triage.** A user reports "my laptop is slow since the update". The first 60 seconds are `Get-ComputerInfo -Property OsName,OsVersion,OsBuildNumber,CsTotalPhysicalMemory` so the ticket records the exact build and memory size.
3. **Kerberos authentication failure.** Domain logons fail with clock-skew errors. `Get-Date` on the client and `w32tm /query /status` reveal a 9-minute drift. On Linux, `timedatectl` shows `System clock synchronized: no`.
4. **Vendor support call.** The vendor asks for the kernel version. `uname -r` gives `6.8.0-48-generic` instantly, without logging in to a GUI.
5. **Documentation and handover.** At the start of a change window an engineer runs `whoami; hostname; date -u` and pastes the output into the change ticket as evidence of who did what, where and when.

Cybersecurity Application

- **Situational awareness after gaining a shell (authorised testing or incident response):** the first commands any responder or penetration tester runs are `whoami`, `hostname`, `id`, `uname -a` (Linux) or `whoami /all`, `hostname`, `$PSVersionTable` (Windows). Knowing the current user, privileges and OS version determines the next steps.
- **Recognising attacker activity:** because attackers run exactly these commands, a SOC analyst seeing `whoami` followed by `whoami /priv` and `net user` executed by a web-server process (for example `w3wp.exe` or `www-data`) treats it as a strong indicator of compromise. You will hunt for such process-creation events on Days 16–17.
- **Time is evidence:** incident timelines need trustworthy timestamps. Recording `date -u` / `Get-Date -Format o` when you collect evidence, and checking that NTP is synchronised, keeps your timeline defensible.
- **PowerShell edition and logging:** Windows PowerShell 5.1 and PowerShell 7 support Script Block Logging, which attackers try to avoid by launching older engines. Knowing how to check `$PSVersionTable` is the first step toward understanding PowerShell security logging (Day 19).

Network Analyst Application

- `hostname -I` (Linux) quickly lists the host's IP addresses before deeper investigation with `ip addr` (Day 11).
- Host names matter for DNS: the name `hostname` prints should resolve to the IP you expect. A mismatch is a classic cause of "it works by IP but not by name" problems (Day 13).
- Accurate time on network devices and hosts is essential for correlating packet captures with firewall and server logs (Days 15 and 27).
- Knowing whether you are in Windows PowerShell 5.1 or PowerShell 7 matters because some networking cmdlets (`Test-Connection` parameters, `Test-NetConnection`) behave differently between them (Day 12).

System Administrator Application

- **Inventory:** `Get-ComputerInfo` and `uname -a` / `/etc/os-release` give OS version and build for patch compliance reporting.
- **Pre-change checklist:** `whoami` (right account and privileges?), `hostname` (right machine?), `date` (inside the approved change window?).

- **Help-first culture:** an administrator who can read `man` and `Get-Help` pages can work on systems without internet access, which is common in secure data centres.
- **Standardising sessions:** administrators set prompts and Windows Terminal profiles so that admin sessions, production hosts and PowerShell editions are visually distinct.

Common Mistakes

Mistake	What happens	Correct approach
Typing `LS` or `Whoami` in Bash	`Command 'LS' not found`	Linux commands are lowercase and case-sensitive
Using `ls -la` in PowerShell 7 on Windows	`Get-ChildItem: A parameter cannot be found that matches parameter name 'la'`	`ls` is an alias for `Get-ChildItem`; use `Get-ChildItem -Force`
Expecting `ls` to be Get-ChildItem on Linux PowerShell	The native `ls` runs and returns text, not objects	Use `Get-ChildItem` in scripts
Running everything as Administrator/ root "to be safe"	Mistakes become system-wide; malware gets full rights	Use a normal shell; elevate only for the specific task
Reading `Get-Help` output and seeing only syntax	Local help files were never downloaded	Run `Update-Help` (admin in 5.1, or `-Scope CurrentUser` in 7) or use `-Online`
Using `Get-ComputerInfo` without `-Property` in a script	Slow (several seconds) and lots of noise	Request only the properties you need
Assuming `uname` shows the Ubuntu version	It shows the **kernel** version	`cat /etc/os-release` or `lsb_release -a`
Using `Get-Date -Format` and then doing date maths	`-Format` returns a **string**; `.AddDays()` fails	Keep the `DateTime` object; format only at the end

Troubleshooting Guide

Symptom	Likely cause	Fix
`bash: <cmd>: command not found`	Typo, wrong case, or package not installed	Check spelling; `type <cmd>`; Ubuntu often suggests `sudo apt install <package>`
`<cmd> : The term '<cmd>' is not recognized as the name of a cmdlet...`	Typo, module not loaded, or command not in PATH	`Get-Command *partial*`; check the edition (5.1 vs 7)
PowerShell window closes immediately when a script is double-clicked	Scripts run in a new window that exits	Run scripts from an open PowerShell session (`.\script.ps1`)
`pwsh` is not recognized	PowerShell 7 not installed	See PowerShell 7 Installation
`man: command not found` or "No manual entry" in a container/WSL minimal image	Ubuntu minimal images strip manuals	`sudo apt install man-db` then `sudo unminimize` (Ubuntu server/minimal)
Prompt shows `#`	You are root	`exit` to return to your normal account unless root is required
`Get-ComputerInfo` not found in PowerShell on Linux	Windows-only cmdlet	Use `uname -a`, `hostnamectl`, `cat /etc/os-release`

Symptom	Likely cause	Fix
Time is wrong	Wrong time zone or NTP not syncing	Linux: `timedatectl`, `sudo timedatectl set-timezone Asia/Kolkata`; Windows: Settings → Time & language, `w32tm /resync` (admin)

Hands-on Lab

Complete **Lab 01: First Contact**. You will open both shells, record a system identity report, discover commands with help, and build your first one-line inventory in each shell. The lab includes verification steps and a full solution.

Summary of tasks:

1. Open Bash and PowerShell; identify the shell edition and version.
2. Record user, host, current directory, OS and kernel/build, architecture and UTC time.
3. Use help to discover the options for long listings and hidden files.
4. Produce a single-line identity summary in each shell.
5. Run the course's `system-info` script and compare it with your manual results.

Practice Questions

1. What is the difference between a terminal and a shell? Give one example of each.
2. Name the two PowerShell editions, their executables and the .NET runtime each uses.
3. What does the `$` at the end of a Bash prompt indicate, and what does `#` indicate?
4. Which command shows the Ubuntu release version, and why is `uname -r` not the answer?
5. Write the PowerShell command that lists only directories in `C:\Windows`.
6. Write the Bash command that lists all files in `/etc` including hidden ones, in long format with human-readable sizes.
7. Why does `ls -la` fail in PowerShell on Windows, and what is the correct PowerShell command?
8. How do you print the current date as `2026-11-08` in Bash and in PowerShell?
9. What PowerShell variable tells you whether you are in Windows PowerShell 5.1 or PowerShell 7?
10. How do you find every PowerShell command whose name contains the word `Service`?
11. Explain why `Get-Date -Format 'yyyy-MM-dd'` followed by `.AddDays(1)` fails.
12. Name three things a security analyst learns from `whoami`, `hostname` and `uname -a` at the start of an investigation.

Practice answers

1. The **terminal** is the window program that displays text and accepts keystrokes (GNOME Terminal, Windows Terminal). The **shell** interprets commands (Bash, PowerShell). One terminal can host many different shells.
2. **Windows PowerShell 5.1**: `powershell.exe`, .NET Framework 4.x, Windows only. **PowerShell 7**: `pwsh`, modern .NET, cross-platform.
3. `$` = normal (unprivileged) user; `#` = root, the superuser.
4. `cat /etc/os-release` (or `lsb_release -a`). `uname -r` prints the **kernel** release (e.g. `6.8.0-48-generic`), not the distribution version.
5. `Get-ChildItem -Path C:\Windows -Directory`

6. `ls -lah /etc`
7. On Windows, `ls` is an alias for `Get-ChildItem`, which has no `-la` parameter. Use `Get-ChildItem -Force` (shows hidden items). The long-format table is the default output.
8. Bash: `date +%F` (or `date +%Y-%m-%d`). PowerShell: `Get-Date -Format yyyy-MM-dd`.
9. `$PSVersionTable` — check `PSEdition` (`Desktop` = 5.1, `Core` = 7) or `PSVersion`.
10. `Get-Command *Service*`
11. `-Format` converts the date to a **string**. Strings do not have an `AddDays()` method. Do the arithmetic first: `(Get-Date).AddDays(1).ToString('yyyy-MM-dd')`.
12. Which account and privilege level they are operating as; which machine (host name) they are on; the OS/kernel version and architecture, which determines relevant vulnerabilities, tools and log locations.

Quiz

Q1. Which component actually executes the system call that reads a file from disk?

- A. The terminal
- B. The shell
- C. The operating system kernel
- D. The prompt

Answer and explanation

C. The terminal only displays text and the shell interprets your command. The kernel performs file, process and device operations on behalf of programs.

Q2. In PowerShell, what does `Get-ChildItem` return?

- A. A block of text
- B. .NET objects such as `FileInfo` and `DirectoryInfo`
- C. A CSV file
- D. Nothing; it only prints to the screen

Answer and explanation

B. PowerShell cmdlets emit objects. The table you see is the default formatting. Because the objects keep properties like `Length` and `LastWriteTime`, you can sort and filter without text parsing.

Q3. You run `ls` in PowerShell 7 on Ubuntu. What runs?

- A. `Get-ChildItem`
- B. `/usr/bin/ls`
- C. An error
- D. `dir`

Answer and explanation

B. The `ls` alias is not defined on non-Windows platforms, so PowerShell runs the native Linux executable, which returns text, not objects.

Q4. Which Bash command prints the kernel release only?

- A. `uname -a`
- B. `uname -m`
- C. `uname -r`
- D. `hostname -r`

Answer and explanation

C. `-r` = kernel release. `-a` prints everything and `-m` prints the architecture. `hostname` has no `-r` option.

Q5. Why should scripts use `Get-ComputerInfo -Property ...` instead of plain `Get-ComputerInfo`?

- A. Plain `Get-ComputerInfo` requires administrator rights
- B. Collecting only required properties is faster and produces cleaner output
- C. Plain `Get-ComputerInfo` only works in PowerShell 7
- D. `-Property` is mandatory

Answer and explanation

B. The cmdlet works without admin rights in 5.1 and 7 on Windows. Gathering all ~180 properties is slow, and specifying `-Property` limits the work and the output.

Q6. What is the safest way to learn about a Bash built-in such as `cd`?

- A. `man cd`
- B. `help cd`
- C. `cd --help` always
- D. `Get-Help cd`

Answer and explanation

B. Built-ins are part of Bash itself and documented by `help`. `man cd` often has no entry, and `cd --help` works in recent Bash versions but `help` is the canonical method.

Q7. Which statement about case sensitivity is correct?

- A. Bash and PowerShell are both case-insensitive
- B. Bash commands are case-sensitive; PowerShell cmdlet names are case-insensitive
- C. PowerShell is case-sensitive; Bash is not
- D. Both are case-sensitive

Answer and explanation

B. `LS` fails in Bash, but `get-childitem` works in PowerShell. Note: Linux **file names** are case-sensitive too, while NTFS is case-insensitive by default.

Q8. You need a timestamp for an incident report that colleagues in three countries will read. Which is best?

- A. `date`
- B. `Get-Date`
- C. `date -u -Iseconds` or `(Get-Date).ToUniversalTime().ToString('o')`
- D. The clock in the taskbar

Answer and explanation

C. ISO 8601 in UTC is unambiguous across time zones and sorts correctly as text.

Homework

1. **Shell inventory (both platforms):** create a text file `day01-inventory.txt` containing the output of the commands from Demo 1, Demo 3 and Demo 4 for each machine you have. In Bash use `command >> day01-inventory.txt`, and in PowerShell use `command | Out-File -Append day01-inventory.txt`. Redirection is taught properly on Day 3. For now, just use it.
2. **Help hunt:** using only built-in help (no web search), find:
   - the `ls` option that sorts by file size,
   - the `Get-ChildItem` parameter that limits recursion depth,
   - the `date` format code for the day of the year,
   - the `Get-Date` property that returns the day of the year.
3. **Alias table:** run `Get-Alias` and write down 10 aliases that match Linux command names. For each, note whether the PowerShell behaviour is truly equivalent (hint: compare `rm -rf` with `Remove-Item`).
4. **Reflection (100 words):** explain to a non-technical manager why the IT team prefers the command line for server administration.

Homework answer key (items 2–3)

- `ls -S` sorts by size (largest first); `ls -lSh` shows sizes too.
- `Get-ChildItem -Depth <n>` (PowerShell 5.0+).
- `date +%j` prints the day of the year (001–366).
- `(Get-Date).DayOfYear`
- Typical aliases: `ls, dir, cd, pwd, cat, cp, mv, rm, rmdir, echo, sleep, sort, kill, ps, man, mount` (some exist only in 5.1, e.g. `curl` and `wget` were aliases for `Invoke-WebRequest` in Windows PowerShell 5.1 but were removed in PowerShell 6+). Behaviour differs: `rm -rf` fails in PowerShell because `-rf` is not a parameter. The equivalent is `Remove-Item -Recurse -Force`.

Professional Tips

- **Put the host name and user in your prompt and window title.** Ubuntu does this by default. In PowerShell, customise the `prompt` function in your `$PROFILE` once you reach Day 23.
- **Learn to read help pages quickly.** Look at *SYNOPSIS/SYNTAX* first, then *EXAMPLES*. Most people jump to examples, which is fine.
- **Prefer full cmdlet names in scripts and aliases at the prompt.** `gci` is fine interactively; `Get-ChildItem` in a script is readable by the next engineer.
- **Use tab completion constantly.** It saves typing and prevents typos in both shells: press Tab to complete paths, commands and PowerShell parameters.
- **Use Windows Terminal** with separate profiles for PowerShell 7, Windows PowerShell 5.1 and Ubuntu (WSL).
- **Always know your edition:** many "PowerShell doesn't work" questions online are really "Windows PowerShell 5.1 vs PowerShell 7" differences.

Command Cheat Sheet

```
# ---- Bash (Ubuntu 24.04) ----
whoami                      # current (effective) user
hostname ; hostname -I      # host name ; all IP addresses
pwd                         # current directory
ls -lah                     # long, all (hidden), human sizes
uname -a ; uname -r         # all kernel info ; kernel release
cat /etc/os-release         # distribution and version
date ; date -u ; date +%F   # local ; UTC ; YYYY-MM-DD
timedatectl                 # time zone and NTP status
man ls ; ls --help          # full manual ; quick help
help cd ; type ls           # built-in help ; what is this name?
```

```
# ---- PowerShell (5.1 / 7.x) ----
whoami ; $env:USERNAME                    # current user
hostname ; $env:COMPUTERNAME              # computer name
Get-Location                              # current location (pwd)
Get-ChildItem -Force                      # list including hidden (ls/dir)
$PSVersionTable                           # PowerShell version and edition
Get-ComputerInfo -Property Os*            # OS details (Windows only)
Get-Date ; Get-Date -Format yyyy-MM-dd    # date object ; formatted string
(Get-Date).ToUniversalTime()              # UTC
Get-Help Get-ChildItem -Examples          # examples
Get-Command *Service* ; Get-Alias ls      # find commands ; resolve alias
```

Completion Checklist

- ☐ I can explain the difference between terminal, shell and kernel.
- ☐ I opened a Bash terminal and a PowerShell session and identified the PowerShell edition.
- ☐ I ran `whoami`, `hostname`, `pwd`, `ls`, `uname` and `date` and understood their output.
- ☐ I ran `Get-Location`, `Get-ChildItem`, `Get-ComputerInfo` and `Get-Date` and understood their output.
- ☐ I can read command syntax (command, options/parameters, arguments; `[optional]`, `<value>`).
- ☐ I used `man`, `--help`, `Get-Help` and `Get-Command` to find information.

- ☐
  I completed Lab 01 and checked it against the solution.
- ☐
  I answered the practice questions and the quiz.
- ☐
  I completed the homework.

Further Reading

- GNU Bash Reference Manual — https://www.gnu.org/software/bash/manual/
- Linux man pages: ls(1), uname(1), date(1), hostname(1), whoami(1), man(1)
- Ubuntu documentation — https://documentation.ubuntu.com/
- What is PowerShell? — https://learn.microsoft.com/en-us/powershell/scripting/overview
- Differences between Windows PowerShell 5.1 and PowerShell 7 — https://learn.microsoft.com/en-us/powershell/scripting/whats-new/differences-from-windows-powershell
- Get-ChildItem, Get-Location, Get-ComputerInfo, Get-Date, Get-Help
- Windows Terminal — https://learn.microsoft.com/en-us/windows/terminal/

Next: Day 02: File and Directory Management

Day 02: File and Directory Management

Lesson overview

Module	1 — Command-Line Fundamentals
Difficulty	Beginner
Estimated time	3–4 hours (75 min reading, 75 min demonstrations, 60–90 min lab and quiz)
Platforms	Ubuntu 24.04 LTS (Bash 5.2, GNU coreutils 9.4) · Windows 11 (Windows PowerShell 5.1 and PowerShell 7.4+)
Lab	Lab 02: Project Workspace

Learning Objectives

By the end of this lesson you will be able to:

1. Describe the **Linux Filesystem Hierarchy** (`/`, `/etc`, `/home`, `/var`, `/usr`, `/tmp`…) and the main **Windows** folders (`C:\Windows`, `C:\Program Files`, `C:\Users`, `C:\ProgramData`).
2. Distinguish **absolute** and **relative** paths, and use `.`, `..`, `~` and `-` correctly.
3. Navigate with `cd` / `Set-Location`, and use `pushd`/`popd` / `Push-Location`/`Pop-Location`.
4. Create, copy, move, rename and delete files and directories with `mkdir`, `touch`, `cp`, `mv`, `rm`, `rmdir` and with `New-Item`, `Copy-Item`, `Move-Item`, `Rename-Item`, `Remove-Item`.
5. Use **wildcards** (`*`, `?`, `[...]`) and understand who expands them in each shell.
6. Perform **recursive** operations deliberately, and apply **safe-deletion practices** (`-i`, `-WhatIf`, `-Confirm`, preview first, quote variables).

Prerequisites

- Day 01 completed: you can open both shells and use `pwd`, `ls`, `Get-Location`, `Get-ChildItem` and help.
- A **non-production** machine or VM. Today you will delete files on purpose, so work only inside the practice folder created in the lab.

Introduction and Concepts

Everything is a file (Linux) and drives plus folders (Windows)

Linux has a **single directory tree** that starts at the root, `/`. Every disk, USB stick, network share and even virtual information about the kernel is *mounted* somewhere inside that tree. There are no drive letters.

Windows uses **drive letters** (`C:\`, `D:\`), each with its own tree. PowerShell generalises this with **PSDrives**: `C:`, `HKLM:` (registry), `Env:` (environment variables), `Cert:` (certificates). `Get-PSDrive` lists them (Day 9 goes deeper).

Diagram — shown rendered on the website; text form:

```
flowchart TD
    R["/ (root)"] --> bin["/bin → /usr/bin<br/>essential commands"]
    R --> etc["/etc<br/>system configuration"]
    R --> home["/home<br/>user home directories"]
    R --> root["/root<br/>root's home"]
    R --> var["/var<br/>variable data: logs, caches, spool"]
    R --> usr["/usr<br/>installed software, libraries"]
    R --> tmp["/tmp<br/>temporary files"]
    R --> dev["/dev<br/>device files"]
    R --> proc["/proc, /sys<br/>virtual kernel information"]
    R --> opt["/opt<br/>optional 3rd-party software"]
    R --> mnt["/mnt, /media<br/>mount points"]
    R --> boot["/boot<br/>kernel and boot loader"]
    home --> st["/home/student"]
    var --> log["/var/log"]
```

The Linux Filesystem Hierarchy Standard (FHS)

Path	Contains	Professional relevance
`/`	Root of the whole tree	Never delete anything here casually
`/bin`, `/sbin`	Essential user/admin programs. On Ubuntu 24.04 these are **symbolic links** to `/usr/bin` and `/usr/sbin` (the "merged /usr" layout)	`ls -ld /bin` shows `/bin -> usr/bin`
`/etc`	System-wide **configuration** text files (`/etc/passwd`, `/etc/hosts`, `/etc/ssh/sshd_config`)	Back up before editing; audited by security teams
`/home`	One directory per user (`/home/student`)	User data; often a separate partition
`/root`	Home of the `root` user	Not readable by normal users
`/var`	Data that changes: `/var/log` (logs), `/var/lib` (state, databases), `/var/cache`, `/var/spool`	Disk-full incidents usually start here (Day 9)
`/tmp`	Temporary files, cleared at boot (and by `systemd-tmpfiles`)	World-writable, so a classic location for malware drops
`/usr`	Installed programs and libraries (`/usr/bin`, `/usr/lib`, `/usr/share`)	Managed by the package manager (Day 10). Do not edit by hand.
`/usr/local`	Software you install manually	Survives package upgrades
`/opt`	Self-contained third-party applications	Vendor software
`/dev`	Device files (`/dev/sda`, `/dev/null`)	Disks appear here (Day 9)
`/proc`, `/sys`	Virtual files generated by the kernel (`/proc/cpuinfo`, `/proc/meminfo`)	Process and hardware investigation (Days 8, 28)
`/boot`	Kernel images and boot loader	A full `/boot` breaks kernel upgrades
`/mnt`, `/media`	Mount points for temporary and removable media	Day 9
`/srv`	Data served by this system (web, FTP)	Optional

Path	Contains	Professional relevance
`/run`	Runtime data since boot (PID files, sockets)	Cleared at boot

Key Windows locations

Path	Contains	PowerShell shortcut
`C:\Windows`	Operating system	`$env:windir`
`C:\Windows\System32`	Core 64-bit system programs and DLLs	
`C:\Program Files`, `C:\Program Files (x86)`	Installed 64-bit / 32-bit applications	`$env:ProgramFiles`
`C:\ProgramData`	Machine-wide application data (hidden)	`$env:ProgramData`
`C:\Users\<name>`	User profile (Desktop, Documents, Downloads)	`$HOME`, `$env:USERPROFILE`, `~`
`C:\Users\<name>\AppData`	Per-user application data: `Roaming`, `Local`, `LocalLow`	`$env:APPDATA`, `$env:LOCALAPPDATA`
`%TEMP%`	Per-user temporary folder	`$env:TEMP`

Absolute and relative paths

An **absolute path** starts from the root and works no matter where you are:

- Linux: `/home/student/projects/report.txt`
- Windows: `C:\Users\student\projects\report.txt`

A **relative path** is interpreted from the **current directory**:

- `projects/report.txt` means "the `projects` folder inside wherever I am now".

Special names:

Symbol	Meaning	Bash	PowerShell
`.`	Current directory	Yes	Yes
`..`	Parent directory	Yes	Yes
`~`	Home directory	Yes (expanded by the shell)	Yes (in file system cmdlets and `cd`). Native programs may not understand it, so use `$HOME`
`-`	Previous directory	`cd -`	`Set-Location -` (PowerShell 6.2+ only)
`/` vs `\`	Separator	`/` only	Both `\` and `/` work in PowerShell

Spaces in names

> Paths containing spaces must be **quoted** in both shells: `cd "My Documents"`, `Set-Location 'C:\Program Files'`. In Bash you can also escape the space: `cd My\ Documents`. Tab completion inserts the quoting for you.

Wildcards (globbing)

Pattern	Matches	Example
`*`	Any number of characters (including none)	`*.log` → `app.log`, `error.log`
`?`	Exactly one character	`file?.txt` → `file1.txt`, `fileA.txt`
`[abc]`	One character from the set	`report[12].csv` → `report1.csv`, `report2.csv`
`[a-z]`	One character from a range	`log[0-9].txt`
`{a,b}`	**Bash brace expansion** (not a wildcard). Generates words even if no file exists	`mkdir {src,docs,tests}`

Who expands the wildcard matters:

- In **Bash**, the *shell* expands `*.log` into a list of matching names **before** the command runs. `rm *.log` actually runs `rm app.log error.log`. If nothing matches, Bash passes the literal text `*.log` (unless `nullglob` is set). Names beginning with `.` are **not** matched by `*`.
- In **PowerShell**, the *cmdlet* expands wildcards in its `-Path` parameter. Use **`-LiteralPath`** when a real file name contains `[` or `]`, because those are wildcard characters (for example `report[final].txt`).

Recursion

"Recursive" means *the directory and everything inside it, at any depth*. It is required to copy or delete a folder tree. In Bash use `cp -r`, `rm -r`, `ls -R`. In PowerShell use `-Recurse`, and limit the depth with `Get-ChildItem -Depth`. Recursion is powerful and dangerous, so always preview before a recursive delete.

Safe deletion: there is no Recycle Bin

`rm` and `Remove-Item` **do not** move files to the Trash or Recycle Bin. Deleted means gone, short of restoring from backup. Professional habits:

1. **Preview** with the same pattern first: `ls *.tmp` before `rm *.tmp`; `Get-ChildItem *.tmp` before `Remove-Item *.tmp`.
2. **Ask for confirmation**: `rm -i`, `rm -I`; `Remove-Item -Confirm`.
3. **Simulate**: `Remove-Item -WhatIf` shows what *would* be deleted without deleting anything.
4. **Quote and guard variables in scripts**: `rm -rf -- "${target:?}/"*` refuses to run when `$target` is empty, which prevents the infamous `rm -rf /*`.
5. **Have a backup** and know where it is before deleting in production.
6. **Never** run recursive forced deletes as root or Administrator unless you have verified the path twice.

Linux Command Reference

Command	Purpose	Key options	Needs root?
`cd`	Change directory (Bash built-in)	`cd`, `cd ~`, `cd -`, `cd ..`	No
`pwd`	Print current directory	`-P`	No
`ls`	List	`-l -a -h -R -d -t`	No
`mkdir`	Make directories	`-p` (parents, no error if exists), `-v`, `-m MODE`	Only outside your areas
`touch`	Create empty file or update timestamps	`-c` (do not create), `-d DATE`	No
`cp`	Copy files/directories	`-r`, `-i`, `-n`, `-v`, `-p`, `-a`, `-u`	No
`mv`	Move or rename	`-i`, `-n`, `-v`, `-u`	No
`rm`	Remove files/directories	`-i`, `-I`, `-r`, `-f`, `-v`, `-d`	No
`rmdir`	Remove **empty** directories	`-p`, `-v`	No
`pushd` / `popd` / `dirs`	Directory stack	`dirs -v`	No
`tree`	Show a directory tree	`-L N`, `-a`, `-d` (install: `sudo apt install tree`)	No
`file`	Identify a file's type by content		No
`stat`	Detailed file metadata (size, inode, times)		No

PowerShell Command Reference

Cmdlet	Aliases (Windows)	Purpose	Key parameters
`Set-Location`	`cd`, `chdir`, `sl`	Change location	`-Path`, `-LiteralPath`, `-` (6.2+)
`Push-Location` / `Pop-Location`	`pushd` / `popd`	Location stack	`-Path`, `-StackName`
`New-Item`	`ni` (`mkdir` is a function)	Create file or directory	`-Path`, `-Name`, `-ItemType File/Directory`, `-Value`, `-Force`
`Copy-Item`	`cp`, `copy`, `cpi`	Copy	`-Path`, `-Destination`, `-Recurse`, `-Force`, `-Filter`, `-WhatIf`
`Move-Item`	`mv`, `move`, `mi`	Move (and rename)	`-Path`, `-Destination`, `-Force`, `-WhatIf`
`Rename-Item`	`ren`, `rni`	Rename in place	`-Path`, `-NewName`
`Remove-Item`	`rm`, `del`, `rmdir`, `rd`, `ri`, `erase`	Delete	`-Path`, `-Recurse`, `-Force`, `-WhatIf`, `-Confirm`, `-LiteralPath`
`Test-Path`	—	Does a path exist? Returns `True/False`	`-Path`, `-PathType Leaf/Container`

Cmdlet	Aliases (Windows)	Purpose	Key parameters
`Get-Item`	`gi`	Get one item (file/folder object)	`-Path`, `-Force`
`Resolve-Path`	`rvpa`	Expand wildcards / relative paths to full paths	`-Path`, `-Relative`
`Join-Path` / `Split-Path`	—	Build / take apart paths safely	`-Parent`, `-Leaf`, `-Extension` (7+)

Detailed Command Explanations

Navigating: `cd` and `Set-Location`

`cd DIR` changes the shell's current directory. `cd` alone (or `cd ~`) goes home, `cd ..` goes up one level, `cd -` returns to the previous directory and prints it. `cd` is a **built-in** because a separate program could not change the shell's own directory.

`Set-Location` does the same in PowerShell and can also move into other providers (`Set-Location HKLM:\SOFTWARE`). Since PowerShell 6.2, `Set-Location -` and `Set-Location +` move backward and forward through location history. In Windows PowerShell 5.1, use `Push-Location`/`Pop-Location` instead.

`pushd DIR` changes directory **and** remembers where you were. `popd` returns. This is great in scripts: `pushd /var/log; ...; popd`.

Creating directories: `mkdir` and `New-Item -ItemType Directory`

`mkdir reports` creates one directory. `mkdir -p projects/2026/q4` creates the whole chain and **does not fail** if it already exists, which makes it safe in scripts. Combine with brace expansion: `mkdir -p lab/{logs,configs,scripts}`.

`New-Item -Path reports -ItemType Directory` creates a directory and returns its `DirectoryInfo` object. Intermediate directories are created automatically. With `-Force`, no error occurs if the directory already exists. On Windows, `mkdir` is a built-in **function** that calls `New-Item -ItemType Directory`. On Linux PowerShell, `mkdir` runs the native program.

Creating files: `touch` and `New-Item -ItemType File`

`touch notes.txt` creates an empty file if it does not exist. If it **does** exist, `touch` only updates its modification time and never changes the content. `touch -c` updates times without creating.

`New-Item -Path notes.txt -ItemType File` creates an empty file. `-Value 'text'` writes initial content.

> **`New-Item -Force` overwrites existing files**
>
> For **files**, `New-Item -Force` replaces an existing file with a new empty (or `-Value`) file, so its content is lost. That is different from `touch`. The safe PowerShell "touch" is:
>
> ```
> if (Test-Path -Path notes.txt) { (Get-Item notes.txt).LastWriteTime = Get-Date }
> else { New-Item -Path notes.txt -ItemType File | Out-Null }
> ```

Copying: `cp` and `Copy-Item`

`cp SOURCE DEST` copies a file. If `DEST` is an existing directory, the file goes inside it with the same name. To copy a **directory** you must add `-r` (recursive). Important options:

- `-i` asks before overwriting; `-n` never overwrites
- `-v` prints each file copied (verbose)
- `-p` preserves mode, ownership and timestamps; `-a` (archive) = `-r` plus preserve everything, including symlinks. Use it for backups.
- `-u` copies only when the source is newer than the destination

`Copy-Item -Path SOURCE -Destination DEST` behaves similarly. `-Recurse` copies folder contents. `-Force` overwrites read-only files and creates hidden items. `-Filter *.log` limits what is copied. `-WhatIf` previews.

> **The nested-folder rule (both shells)**
>
> `cp -r src backup` and `Copy-Item src backup -Recurse` produce **`backup/` = copy of `src`** if `backup` does **not** exist, but **`backup/src/`** if `backup` already exists. To copy only the *contents*, use `cp -r src/. backup/` or `Copy-Item src\* backup -Recurse`.

Moving and renaming: `mv`, `Move-Item`, `Rename-Item`

In Linux, moving and renaming are the same operation: `mv old.txt new.txt` renames, `mv file.txt archive/` moves. Within one filesystem a move is instant (only the directory entry changes); across filesystems it copies and then deletes. `-i` prompts before overwrite, `-n` never overwrites, `-v` is verbose.

`Move-Item` moves and can rename at the same time. `Rename-Item -Path old.txt -NewName new.txt` renames in place. `-NewName` takes **only a name, not a path**. By default `Move-Item` refuses to overwrite an existing destination file. Add `-Force` to allow it.

Deleting: `rm`, `rmdir`, `Remove-Item`

- `rm file` deletes files. `rm -r dir` deletes a directory tree. `rm -f` never prompts and ignores missing files.
- `rm -i` asks for every file; `rm -I` asks **once** when deleting more than three files or recursively, which is a sensible default.
- `rmdir dir` removes a directory **only if it is empty**, which makes it a safe way to clean up.
- `Remove-Item path` deletes. Non-empty folders need `-Recurse` (otherwise PowerShell asks for confirmation). Hidden or read-only items need `-Force`. `-WhatIf` simulates and `-Confirm` prompts.

Every PowerShell cmdlet that changes the system supports `-WhatIf` and `-Confirm` (the *ShouldProcess* mechanism). Use them every time you are unsure.

Helpers: `Test-Path`, `Join-Path`, `stat`, `file`

- `Test-Path C:\Temp\lab` returns `True` or `False`. It is essential in scripts before creating or deleting.
- `Join-Path $HOME 'lab'` builds a path with the correct separator for the OS. This is better than string concatenation.
- `stat file` (Linux) shows size, permissions, inode and the **access/modify/change** times.

- `file mystery.bin` identifies content regardless of the extension. Attackers rename executables to `.jpg`, but `file` is not fooled.

Syntax and Parameters

Bash

```
cd [-L|-P] [DIR | - | ~]
mkdir [-p] [-v] [-m MODE] DIR...
touch [-c] [-d DATE] FILE...
cp [-r] [-i|-n] [-v] [-p|-a] [-u] SOURCE... DEST
mv [-i|-n] [-v] [-u] SOURCE... DEST
rm [-i|-I] [-r] [-f] [-v] [--] FILE...
rmdir [-p] [-v] DIR...
```

Option	Applies to	Meaning
`-p`	mkdir	Create parent directories as needed; no error if existing
`-p`	cp	Preserve mode, ownership, timestamps
`-a`	cp	Archive: recursive + preserve all attributes + copy symlinks as symlinks
`-r` / `-R`	cp, rm	Recursive
`-i`	cp, mv, rm	Interactive: prompt before each overwrite/removal
`-I`	rm	Prompt once before removing more than 3 files or recursively
`-n`	cp, mv	No-clobber: never overwrite
`-f`	rm	Force: never prompt, ignore nonexistent files
`-u`	cp, mv	Update: only when source is newer or destination missing
`-v`	most	Verbose: print what is done
`--`	rm, cp, mv	End of options: needed for names that start with `-` (e.g. `rm -- -weird.txt`)

PowerShell

```
Set-Location [[-Path] <String>] [-LiteralPath <String>] [-PassThru]
New-Item [-Path] <String[]> [-ItemType <String>] [-Name <String>] [-Value <Object>] [-Force] [-
WhatIf] [-Confirm]
Copy-Item [-Path] <String[]> [[-Destination] <String>] [-Recurse] [-Force] [-Filter <String>]
          [-Include <String[]>] [-Exclude <String[]>] [-PassThru] [-WhatIf] [-Confirm]
Move-Item [-Path] <String[]> [[-Destination] <String>] [-Force] [-PassThru] [-WhatIf] [-Confirm]
Rename-Item [-Path] <String> [-NewName] <String> [-Force] [-PassThru] [-WhatIf] [-Confirm]
Remove-Item [-Path] <String[]> [-LiteralPath <String[]>] [-Recurse] [-Force] [-Filter <String>]
            [-Include <String[]>] [-Exclude <String[]>] [-WhatIf] [-Confirm]
Test-Path [-Path] <String[]> [-PathType <Any|Container|Leaf>]
```

Parameter	Meaning
`-ItemType`	`File`, `Directory`, `SymbolicLink`, `HardLink`, `Junction`
`-Force`	Allow hidden/read-only items; **New-Item:** overwrite existing files; **Move-Item:** overwrite destination
`-Recurse`	Include all child items
`-Filter`	Fast provider filter (one pattern)
`-Include` / `-Exclude`	Additional patterns. With `Copy-Item`/`Remove-Item` they work most reliably together with `-Recurse` or a wildcard path
`-PassThru`	Output the resulting object (useful for verification)
`-WhatIf`	Show what would happen; do nothing
`-Confirm`	Ask before each action
`-LiteralPath`	Treat the path exactly as typed (no wildcard interpretation)

Practical Demonstrations

All demos work inside a disposable practice folder in your home directory. Outputs are **examples**: names match the commands, while dates, users and sizes will differ on your system.

Demo 1 — Explore the hierarchy

Bash

```
cd /
ls
ls -ld /bin /sbin
cd /var/log
pwd
cd ..
pwd
cd -
cd ~
pwd
```

Expected output (example, Ubuntu 24.04):

```
bin  bin.usr-is-merged  boot  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc
root  run  sbin  snap  srv  sys  tmp  usr  var
lrwxrwxrwx 1 root root 7 Apr 22  2024 /bin -> usr/bin
lrwxrwxrwx 1 root root 8 Apr 22  2024 /sbin -> usr/sbin
/var/log
/var
/var/log
/home/student
```

PowerShell

```
Set-Location C:\
Get-ChildItem -Directory | Select-Object Name
Set-Location $env:windir
Get-Location
Set-Location ..
Get-Location
```

```
Set-Location ~
Get-Location
$env:TEMP
```

Expected output (example):

```
Name
----
PerfLogs
Program Files
Program Files (x86)
Users
Windows

Path
----
C:\Windows

Path
----
C:\

Path
----
C:\Users\student

C:\Users\student\AppData\Local\Temp
```

Demo 2 — Build a project structure

Bash

```
mkdir -p ~/day02/project/{docs,src,logs,backup}
cd ~/day02/project
touch docs/readme.txt src/app.sh logs/app.log logs/error.log logs/debug.tmp
ls -R
```

Expected output:

```
.:
backup  docs  logs  src

./backup:

./docs:
readme.txt

./logs:
app.log  debug.tmp  error.log

./src:
app.sh
```

PowerShell

```
$root = Join-Path $HOME 'day02\project'
'docs','src','logs','backup' | ForEach-Object { New-Item -Path (Join-Path $root $_) -ItemType
Directory -Force | Out-Null }
Set-Location $root
'docs\readme.txt','src\app.ps1','logs\app.log','logs\error.log','logs\debug.tmp' |
```

```
    ForEach-Object { New-Item -Path $_ -ItemType File | Out-Null }
Get-ChildItem -Recurse -File | Select-Object FullName
```

Expected output (example):

```
FullName
--------
C:\Users\student\day02\project\docs\readme.txt
C:\Users\student\day02\project\logs\app.log
C:\Users\student\day02\project\logs\debug.tmp
C:\Users\student\day02\project\logs\error.log
C:\Users\student\day02\project\src\app.ps1
```

`ForEach-Object` runs the block once per input item, and `$_` is the current item. It is explained fully on Day 5. `| Out-Null` discards the objects `New-Item` would otherwise print.

Demo 3 — Copy, including the nested-folder rule

Bash

```
cp -v logs/*.log backup/
cp -r docs docs-copy          # docs-copy did not exist → becomes a copy of docs
cp -r docs docs-copy          # now it exists → creates docs-copy/docs
ls -R docs-copy
```

Expected output:

```
'logs/app.log' -> 'backup/app.log'
'logs/error.log' -> 'backup/error.log'
docs-copy:
docs  readme.txt

docs-copy/docs:
readme.txt
```

PowerShell

```
Copy-Item -Path logs\*.log -Destination backup -PassThru | Select-Object Name
Copy-Item -Path docs -Destination docs-copy -Recurse
Copy-Item -Path docs -Destination docs-copy -Recurse
Get-ChildItem docs-copy -Recurse | Select-Object FullName
```

Expected output (example):

```
Name
----
app.log
error.log

FullName
--------
C:\Users\student\day02\project\docs-copy\docs
C:\Users\student\day02\project\docs-copy\readme.txt
C:\Users\student\day02\project\docs-copy\docs\readme.txt
```

Demo 4 — Move and rename

Bash

```
mv -v src/app.sh src/main.sh
mkdir -p archive
mv -v logs/*.tmp archive/
mv -i backup/app.log logs/app.log    # prompts because logs/app.log exists
```

Expected output:

```
renamed 'src/app.sh' -> 'src/main.sh'
renamed 'logs/debug.tmp' -> 'archive/debug.tmp'
mv: overwrite 'logs/app.log'? n
```

PowerShell

```
Rename-Item -Path src\app.ps1 -NewName main.ps1 -PassThru | Select-Object Name
New-Item -Path archive -ItemType Directory -Force | Out-Null
Move-Item -Path logs\*.tmp -Destination archive -PassThru | Select-Object FullName
Move-Item -Path backup\app.log -Destination logs\app.log
```

Expected output (example):

```
Name
----
main.ps1

FullName
--------
C:\Users\student\day02\project\archive\debug.tmp

Move-Item : Cannot create a file when that file already exists.
```

The last command fails **on purpose**: `Move-Item` will not overwrite an existing file unless you add `-Force`.

Demo 5 — Safe deletion

Bash

```
ls logs/*.log               # 1. preview exactly what the pattern matches
rm -i logs/error.log        # 2. confirm a single delete
rmdir archive               # 3. fails: not empty, so nothing is lost
rm -rI docs-copy            # 4. one confirmation for a recursive delete
target=""                   # 5. guard against empty variables in scripts
rm -rf -- "${target:?target is empty}/"
```

Expected output:

```
logs/app.log  logs/error.log
rm: remove regular empty file 'logs/error.log'? y
rmdir: failed to remove 'archive': Directory not empty
```

```
rm: remove 1 argument recursively? y
bash: target: target is empty
```

The `${target:?message}` expansion stops the command with an error when the variable is empty or unset, so `rm` never runs.

PowerShell

```powershell
Get-ChildItem logs\*.log                    # 1. preview
Remove-Item logs\error.log -WhatIf          # 2. simulate
Remove-Item logs\error.log -Confirm         # 3. confirm
Remove-Item docs-copy -Recurse -WhatIf      # 4. preview recursive delete
Remove-Item docs-copy -Recurse
$target = ''
if ([string]::IsNullOrWhiteSpace($target)) { Write-Warning 'Target is empty - refusing to
delete.' }
```

Expected output (example):

```
    Directory: C:\Users\student\day02\project\logs

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        08-11-2026     16:02              0 app.log
-a----        08-11-2026     16:02              0 error.log

What if: Performing the operation "Remove File" on target "C:
\Users\student\day02\project\logs\error.log".

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:
\Users\student\day02\project\logs\error.log".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

What if: Performing the operation "Remove Directory" on target "C:
\Users\student\day02\project\docs-copy".
WARNING: Target is empty - refusing to delete.
```

Demo 6 — Wildcards and literal paths

Bash

```bash
cd ~/day02/project
touch file1.txt file2.txt fileA.txt file10.txt .hidden.txt 'report[final].txt'
ls file?.txt
ls file[0-9].txt
ls *.txt                # .hidden.txt is NOT matched
ls -d .*.txt
echo {jan,feb,mar}-report.csv
```

Expected output:

```
file1.txt  file2.txt  fileA.txt
file1.txt  file2.txt
file10.txt  file1.txt  file2.txt  fileA.txt  'report[final].txt'
.hidden.txt
jan-report.csv feb-report.csv mar-report.csv
```

PowerShell

```powershell
'file1.txt','file2.txt','fileA.txt','file10.txt','report[final].txt' |
    ForEach-Object { New-Item -Path . -Name $_ -ItemType File -Force | Out-Null }
Get-ChildItem file?.txt -Name
Get-ChildItem 'file[0-9].txt' -Name
Get-Item 'report[final].txt'           # wildcard interpretation: returns nothing
Get-Item -LiteralPath 'report[final].txt' | Select-Object Name
```

Expected output (example):

```
file1.txt
file2.txt
fileA.txt
file1.txt
file2.txt

Name
----
report[final].txt
```

`New-Item -Name` treats the name literally, so it can create `report[final].txt`. `Get-Item 'report[final].txt'` interprets `[final]` as "one of f, i, n, a, l", finds no match and returns nothing. `-LiteralPath` fixes it.

Linux vs PowerShell Comparison

Task	Bash	PowerShell	Notes
Go home	`cd` · `cd ~`	`Set-Location ~` · `cd ~`	
Up one level	`cd ..`	`Set-Location ..`	
Previous directory	`cd -`	`Set-Location -` (6.2+) · `Pop-Location`	Not in 5.1
Make directory (with parents)	`mkdir -p a/b/c`	`New-Item a\b\c -ItemType Directory -Force`	PowerShell creates parents automatically
Create empty file	`touch f.txt`	`New-Item f.txt -ItemType File`	`New-Item -Force` overwrites; `touch` does not
Update timestamp	`touch f.txt`	`(Get-Item f.txt).LastWriteTime = Get-Date`	
Copy file	`cp a b`	`Copy-Item a b`	
Copy directory	`cp -r src dst`	`Copy-Item src dst -Recurse`	Same nesting rule
Copy preserving attributes	`cp -a`	`robocopy src dst /E /COPY:DAT`	`Copy-Item` does not preserve ACLs; robocopy is the Windows tool for that
Move / rename	`mv a b`	`Move-Item a b` · `Rename-Item a -NewName b`	
No-overwrite	`cp -n`, `mv -n`	default for `Move-Item`; `Copy-Item` overwrites unless you test first	Check with `Test-Path`

Task	Bash	PowerShell	Notes
Delete file	`rm f`	`Remove-Item f`	No recycle bin in either
Delete tree	`rm -r d`	`Remove-Item d -Recurse`	
Preview a delete	`ls pattern` first	`Remove-Item ... -WhatIf`	`-WhatIf` is a real simulation
Confirm	`rm -i` / `rm -I`	`-Confirm`	
Remove empty dir only	`rmdir d`	`Remove-Item d` (prompts if not empty)	
Does it exist?	`[ -e path ]` · `test -d dir`	`Test-Path path`	Bash tests are covered on Day 22
Literal name with `[ ]`	quote it: `'a[1].txt'`	`-LiteralPath`	
Brace expansion	`mkdir {a,b,c}`	`'a','b','c' | % { mkdir $_ }`	No braces in PowerShell

Real-World IT Scenarios

1. **Standard server layout.** A team requires every application server to have `/opt/app/{bin,conf,logs,data}`. One `mkdir -p` line in a build script creates it identically on every host, and running it twice is harmless.
2. **Pre-change backup.** Before editing `/etc/ssh/sshd_config`, an admin runs `sudo cp -a /etc/ssh/sshd_config /etc/ssh/sshd_config.bak.$(date +%F)`. If the change breaks SSH, the original is one `cp` away. Windows equivalent: `Copy-Item C:\App\config.xml "C:\App\config.xml.bak.$(Get-Date -Format yyyy-MM-dd)"`.
3. **User profile migration.** A help-desk technician moves a departing employee's documents to an archive share with `Move-Item` / `robocopy`, keeping the original folder names.
4. **Cleaning old build artefacts.** A developer deletes `*.tmp` files from a build tree. They preview with `Get-ChildItem -Recurse -Filter *.tmp` first, then pipe the same command to `Remove-Item -WhatIf`, and only then delete.
5. **The empty-variable disaster.** A cleanup script ran `rm -rf $DIR/*` with `$DIR` unset and deleted the system root's contents. Quoting plus `${DIR:?}` would have stopped it. This is a real, recurring class of production incident.

Cybersecurity Application

- **Know where attackers hide files:** `/tmp`, `/var/tmp` and `/dev/shm` on Linux (world-writable), and `%TEMP%`, `%APPDATA%`, `C:\ProgramData` and `C:\Users\Public` on Windows are favourite drop locations for malware and tools. You will search them by modification time on Day 4.
- **Hidden files:** names starting with `.` on Linux (`ls -a`) and the Hidden attribute on Windows (`Get-ChildItem -Force`) are a trivial but common way to hide persistence scripts.
- **File type vs extension:** `file invoice.pdf` revealing `ELF 64-bit LSB executable` is a red flag. On Windows, check the first bytes: `Get-Content invoice.pdf -Encoding Byte -TotalCount 2` in 5.1 (or `-AsByteStream` in 7) returning `77 90` ("MZ") means a Windows executable.
- **Evidence preservation:** during incident response, **copy** suspicious files with timestamps preserved (`cp -a`, `robocopy /COPY:DAT`) instead of moving or opening them, and never delete them. Changing timestamps destroys timeline evidence.
- **Ransomware awareness:** recursive operations across shares are exactly what ransomware does at scale. Unexpected mass renames or deletions are a key detection signal.

Network Analyst Application

- Network configuration lives in files: `/etc/hosts`, `/etc/resolv.conf`, `/etc/netplan/*.yaml` on Ubuntu, and `C:\Windows\System32\drivers\etc\hosts` on Windows. Navigating to them and **backing them up before editing** is a daily task (Days 11–13).
- Packet captures (`.pcap`) and device configuration backups pile up quickly. Organise them by date with `mkdir -p captures/$(date +%F)` and clean them with previewed deletes.
- Copying configs from many devices into one structured folder tree (`site/device/date`) makes later comparison easy.

System Administrator Application

- **Idempotent setup:** `mkdir -p` and `New-Item -Force` (for directories) let setup scripts run repeatedly without errors.
- **Change management:** a timestamped `.bak` copy before every configuration edit, using `cp -a` or `Copy-Item`.
- **Disk housekeeping:** previewed, pattern-based cleanups of `/var/tmp`, old logs and temp folders (Day 9 adds size analysis).
- **Path hygiene in scripts:** use `Join-Path` and quoted `"$var"`, so scripts work with spaces and on every OS.

Common Mistakes

Mistake	What happens	Correct approach
`rm -rf $DIR/*` with an empty `$DIR`	Deletes everything under `/` that you have rights to	`rm -rf -- "${DIR:?}"/*` and never run as root casually
`rm -rf` in PowerShell	Error: parameter `rf` not found	`Remove-Item -Recurse -Force`
Expecting deleted files in the Recycle Bin	They are gone permanently	Back up first; use `-WhatIf` / `-i`
`New-Item -Force file.txt` used like `touch`	Existing content wiped	`Test-Path` first, or update `LastWriteTime`
`cp -r src dst` run twice	Second run creates `dst/src`	Copy contents: `cp -r src/. dst/` / `Copy-Item src\* dst -Recurse`
Forgetting quotes around paths with spaces	`cd: too many arguments` / wrong item	Quote: `"My Folder"`
Using `Get-Item` on a file name containing `[ ]`	Nothing returned or wrong file	`-LiteralPath`
`Rename-Item a.txt -NewName folder\b.txt`	Error: `-NewName` takes only a name	Use `Move-Item` to change folders
`rmdir` on a non-empty directory in Bash	`Directory not empty`	Intended safety; use `rm -r` only after review
Working in `C:\Windows\System32` by accident (elevated shells start there)	Files created or deleted in a system folder	`Set-Location ~` first; check `Get-Location`

Troubleshooting Guide

Error message	Cause	Fix
`mkdir: cannot create directory 'x': File exists`	Already exists	Use `mkdir -p`
`mkdir: cannot create directory '/opt/app': Permission denied`	Normal users cannot write to `/opt`	`sudo mkdir -p /opt/app` if appropriate, or work in `~`
`cp: -r not specified; omitting directory 'docs'`	Copying a directory without recursion	`cp -r docs dest`
`rm: cannot remove 'x': Is a directory`	Directory needs `-r` (or `rmdir` if empty)	`rmdir x` / `rm -r x`
`rm: cannot remove 'x': Permission denied`	No write permission on the **parent directory**	Check `ls -ld .` (Day 7)
`New-Item : An item with the specified name ... already exists.`	Already exists	Add `-Force` (directories) or `Test-Path` first
`Remove-Item : ... The process cannot access the file because it is being used by another process.`	File open in an application	Close the app; find the locking process (Day 8)
`Remove-Item : ... is an item with children and the Recurse parameter was not specified`	Non-interactive delete of a non-empty folder	Add `-Recurse` after a `-WhatIf` preview
`Copy-Item : Access to the path ... is denied.`	Read-only, system or permission problem	`-Force` for read-only items; run elevated only if it is truly a system path
`Set-Location : Cannot find path ... because it does not exist.`	Typo or relative path from the wrong location	`Get-Location`; use tab completion; use an absolute path
`bash: cd: too many arguments`	Unquoted space in the path	Quote the path

Hands-on Lab

Complete **Lab 02: Project Workspace**. You will build a standard project folder tree, populate it, back it up correctly (avoiding the nested-folder trap), reorganise files with wildcards, and safely clean up using preview and simulation. The lab includes verification commands and a full solution for both shells.

Practice Questions

1. What is the difference between `/etc` and `/var` on Linux? Give one example file from each.
2. Why are `/bin` and `/sbin` symbolic links on Ubuntu 24.04?
3. You are in `/home/student/projects`. Write the absolute and a relative path to `/home/student/notes.txt`.
4. Write one Bash command that creates `lab/logs`, `lab/configs` and `lab/scripts` at once, even if `lab` does not exist.
5. Write the PowerShell equivalent of question 4.
6. What is the danger of `New-Item -Path data.csv -ItemType File -Force` when `data.csv` already exists?
7. Explain what happens when you run `cp -r src backup` twice.

8. Which `rm` option asks only once before a recursive delete?
9. How do you see what `Remove-Item C:\Temp\old -Recurse` would delete without deleting anything?
10. Why does `ls *` not show `.bashrc`, and how do you include it?
11. A file is named `log[2026].txt`. Why does `Get-Item log[2026].txt` fail to find it, and what is the fix?
12. Write a Bash command that deletes the directory stored in `$target` but refuses to run if `$target` is empty.
13. Which PowerShell cmdlet renames a file without moving it, and which parameter takes the new name?
14. What does `cd -` do, and what is the PowerShell equivalent in 5.1 and in 7?

Practice answers

1. `/etc` holds **configuration** (e.g. `/etc/hosts`, `/etc/passwd`). `/var` holds **changing data** (e.g. `/var/log/syslog`, `/var/log/auth.log`).
2. Ubuntu uses the **merged /usr** layout: all programs live in `/usr/bin` and `/usr/sbin`, and the top-level directories remain as compatibility symlinks.
3. Absolute: `/home/student/notes.txt`. Relative: `../notes.txt` (or `~/notes.txt`, which is home-relative).
4. `mkdir -p lab/{logs,configs,scripts}`
5. `'logs','configs','scripts' | ForEach-Object { New-Item -Path "lab\$_" -ItemType Directory -Force }`
6. `-Force` **replaces** the existing file with a new empty one, so all its data is lost.
7. First run: `backup` does not exist, so it becomes a copy of `src`. Second run: `backup` exists, so `src` is copied **inside** it as `backup/src`.
8. `rm -I`
9. Add `-WhatIf`: `Remove-Item C:\Temp\old -Recurse -WhatIf`
10. Bash wildcards do not match names starting with `.`. Use `ls -a`, or `ls -d .*` for hidden names only (or `shopt -s dotglob`).
11. `[2026]` is treated as a wildcard character set (one of 2, 0, 6). Use `Get-Item -LiteralPath 'log[2026].txt'`.
12. `rm -rf -- "${target:?target is empty}"`
13. `Rename-Item -Path old.txt -NewName new.txt`. `-NewName` takes the new name only.
14. `cd -` returns to the previous directory and prints it. PowerShell 7 (6.2+): `Set-Location -`. In 5.1: use `Push-Location` before moving and `Pop-Location` to return.

Quiz

Q1. Which directory stores system-wide configuration files on Linux?

- A. `/usr`
- B. `/etc`
- C. `/var`
- D. `/opt`

Answer and explanation

B. `/etc` holds configuration. `/usr` holds installed software, `/var` changing data such as logs, and `/opt` add-on applications.

Q2. `cp -r reports archive` is run and `archive` already exists. Where do the files end up?

- A. Directly in `archive/`
- B. In `archive/reports/`
- C. The command fails
- D. `archive` is replaced

Answer and explanation

B. When the destination directory exists, the source directory is copied inside it. PowerShell `Copy-Item -Recurse` behaves the same way.

Q3. Which PowerShell parameter **simulates** a destructive command?

- A. `-Force`
- B. `-Confirm`
- C. `-WhatIf`
- D. `-PassThru`

Answer and explanation

C. `-WhatIf` reports what would happen and changes nothing. `-Confirm` asks interactively but does act if you answer Yes.

Q4. What does `mkdir -p /srv/app/logs` do if `/srv/app/logs` already exists?

- A. Fails with "File exists"
- B. Deletes and recreates it
- C. Succeeds silently
- D. Creates `/srv/app/logs/logs`

Answer and explanation

C. `-p` makes `mkdir` idempotent: no error if the directory exists, and parents are created as needed.

Q5. Which statement about deleting is true?

- A. `rm` moves files to the Trash
- B. `Remove-Item` moves files to the Recycle Bin
- C. Both delete permanently
- D. Both ask for confirmation by default for every file

Answer and explanation

C. Neither uses the Trash or Recycle Bin. `Remove-Item` prompts only for non-empty folders without `-Recurse`, and `rm` prompts only with `-i`/`-I` (or for write-protected files).

Q6. In Bash, who expands `*.log` in `rm *.log`?

- A. The `rm` program
- B. The kernel
- C. The Bash shell, before `rm` runs
- D. The terminal

Answer and explanation

C. Bash replaces the pattern with matching names and then starts `rm` with that list. In PowerShell, by contrast, the cmdlet expands wildcards in `-Path`.

Q7. Which command safely creates `notes.txt` in PowerShell **without** destroying it if it already exists?

- A. `New-Item notes.txt -ItemType File -Force`
- B. `if (-not (Test-Path notes.txt)) { New-Item notes.txt -ItemType File }`
- C. `Remove-Item notes.txt; New-Item notes.txt`
- D. `Set-Content notes.txt ''`

Answer and explanation

B. It creates the file only if it is missing. A and D overwrite the content, and C deletes it first.

Q8. Why should production scripts use `"${dir:?}"` rather than `$dir` in `rm -rf` commands?

- A. It is faster
- B. It aborts if `dir` is empty or unset, preventing deletion of the wrong path
- C. It enables recursion
- D. It makes `rm` interactive

Answer and explanation

B. Parameter expansion `${var:?}` causes an error and stops the command when the variable is empty. The quotes also protect paths containing spaces.

Homework

1. **Map the filesystems:** for each of 10 FHS directories, write one sentence on what it contains and run `ls -ld` on it. Do the same for 6 Windows folders using `Get-Item`, and note the `Mode` attributes.
2. **Rebuild the lab without notes:** delete your Lab 02 workspace (preview first!) and rebuild it from memory in **both** shells.
3. **Write a safe cleanup one-liner** in each shell that deletes `*.tmp` files under `~/day02` recursively, with a preview step. Bash hint: `find` (Day 4) or `shopt -s globstar` and `**/*.tmp`.

4. **Research:** read `man cp` and `Get-Help Copy-Item -Full`. List three options or parameters not covered today and what they do.

Homework answer key (item 3)

```
shopt -s globstar nullglob
ls ~/day02/**/*.tmp              # preview
rm -iv ~/day02/**/*.tmp          # confirm each
```

```
$tmp = Get-ChildItem -Path "$HOME\day02" -Filter *.tmp -Recurse -File
$tmp | Select-Object FullName            # preview
$tmp | Remove-Item -WhatIf               # simulate
$tmp | Remove-Item                       # delete
```

Professional Tips

- **Tab-complete every path.** It eliminates typos and handles spaces and quoting automatically.
- **Make `rm` safer interactively:** some admins add `alias rm='rm -I'` to `~/.bashrc`. Never rely on aliases inside scripts, though; be explicit.
- **Use `-WhatIf` reflexively** with `Remove-Item`, `Move-Item` and every new cmdlet that changes things. In a script, `$WhatIfPreference = $true` turns it on for everything.
- **Timestamp your backups:** `cp -a file{,.bak.$(date +%F-%H%M)}` uses brace expansion to create `file.bak.2026-11-08-1602`.
- **Prefer `robocopy` on Windows for large or permission-sensitive copies.** It can resume, retry, log and preserve ACLs (`robocopy src dst /E /COPY:DATSOU /R:2 /W:5 /LOG:copy.log`).
- **Keep work out of system folders:** start every session with `cd ~` / `Set-Location ~` until you deliberately need elsewhere.

Command Cheat Sheet

```
# ---- Bash ----
cd /path ; cd .. ; cd ~ ; cd -          # navigate
pushd /var/log ; popd                    # directory stack
mkdir -p a/b/{c,d}                       # nested dirs, idempotent
touch file.txt                           # create / update timestamp
cp -v a.txt b.txt ; cp -a src/ dst/      # copy ; archive copy
cp -r src/. dst/                         # copy contents only
mv -i old new ; mv *.log archive/        # rename ; move
rm -i f ; rm -rI dir ; rmdir emptydir    # safe deletes
rm -rf -- "${dir:?}"                     # guarded script delete
ls -d .* ; file x ; stat x               # hidden ; type ; metadata
```

```
# ---- PowerShell ----
Set-Location C:\path ; Set-Location ..   # navigate
Push-Location C:\Windows ; Pop-Location  # location stack
New-Item a\b\c -ItemType Directory -Force
New-Item file.txt -ItemType File         # careful: -Force overwrites files
Copy-Item a.txt b.txt ; Copy-Item src dst -Recurse
Copy-Item src\* dst -Recurse             # copy contents only
Move-Item old.txt archive\ ; Rename-Item a.txt -NewName b.txt
Remove-Item f -WhatIf ; Remove-Item dir -Recurse -Confirm
Test-Path C:\x ; Join-Path $HOME lab ; Get-Item -LiteralPath 'a[1].txt'
```

Completion Checklist

- ☐ I can name the purpose of `/etc`, `/home`, `/var`, `/usr`, `/tmp`, `/opt`, `/proc` and `/dev`.
- ☐ I know the main Windows folders and their `$env:` shortcuts.
- ☐ I can use absolute and relative paths, `.`, `..`, `~` and `-`.
- ☐ I can create, copy, move, rename and delete files and folders in both shells.
- ☐ I understand the nested-folder rule for recursive copies.
- ☐ I can use `*`, `?`, `[...]` and Bash brace expansion, and I know when to use `-LiteralPath`.
- ☐ I always preview deletions and know `-i`, `-I`, `-WhatIf` and `-Confirm`.
- ☐ I completed Lab 02 and checked it against the solution.
- ☐ I answered the practice questions and quiz.
- ☐ I completed the homework.

Further Reading

- Filesystem Hierarchy Standard 3.0 — https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html
- `man hier` — https://man7.org/linux/man-pages/man7/hier.7.html
- Linux man pages: cp(1), mv(1), rm(1), mkdir(1), touch(1), glob(7)
- Bash manual: Filename Expansion and Brace Expansion — https://www.gnu.org/software/bash/manual/bash.html#Shell-Expansions
- PowerShell: New-Item, Copy-Item, Move-Item, Remove-Item, Set-Location, about_Wildcards
- robocopy — https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/robocopy

---

Previous: Day 01: Introduction to Linux and PowerShell · **Next:** Day 03: Reading and Managing Files (see the release schedule)

Path A — Cybersecurity / SOC Analyst

A Security Operations Centre (SOC) analyst monitors alerts, investigates suspicious activity, and escalates or contains incidents. The command line lets you verify an alert **on the host itself** when the SIEM view is not enough.

Competencies and where they are taught

Competency	Linux skills	Windows skills	Days
Situational awareness on a host	`whoami`, `id`, `hostname`, `uname -a`	`whoami /all`, `Get-ComputerInfo`	1, 6
Hidden and suspicious files	`ls -la`, `find -mtime`, `file`	`Get-ChildItem -Force`, `Get-FileHash`	2, 4, 19
Log searching	`grep`, `journalctl`, `/var/log/auth.log`	`Get-WinEvent -FilterHashtable`, `Select-String`	3, 4, 16
Linux authentication monitoring	`last`, `lastb`, `who`, sshd logs	—	17
Windows security event analysis	—	Event IDs 4624, 4625, 4634, 4672, 4720, 4740	17
Failed-login detection	`grep "Failed password"` + `awk/sort/uniq -c`	`Get-WinEvent` 4625 + `Group-Object`	17, 24
Process investigation	`ps -ef --forest`, `/proc/<pid>`	`Get-Process`, `Get-CimInstance Win32_Process` (parent PID, command line)	8, 28
Network connection investigation	`ss -tunap`, `lsof -i`	`Get-NetTCPConnection` + `OwningProcess`	14, 20
Firewall auditing	`ufw status verbose`, `nft list ruleset`	`Get-NetFirewallProfile`, `Get-NetFirewallRule`	18
Security baseline	sudoers, SUID files, world-writable files	Execution policy, Defender status, local admins	19
Suspicious activity investigation	`tcpdump`, Nmap against own hosts	`pktmon`, `Get-NetTCPConnection`	20, 27
Security reporting	Text/CSV reports	`Export-Csv`, `ConvertTo-Json`	24, 29

SOC-style case studies (released with the module)

1. **Brute force against SSH** (Day 17): hundreds of `Failed password` entries from one IP. Count, timeline, was any attempt successful?
2. **Password spraying on Windows** (Day 17): many 4625 events across many accounts from one source. Which accounts were targeted?
3. **Web shell indicator** (Days 8, 20): `www-data` spawning `bash` with an outbound connection. Parent/child processes and the socket owner.

4. **Unexpected listening port** (Days 14, 20): a new service on port 4444. Which process and file are behind it, and what is its hash?
5. **Firewall drift** (Day 18): an inbound "Allow Any" rule appeared. When, which profile, and how do you safely revert it?

Final project: Security Investigation Toolkit

Build a pair of scripts (`soc-toolkit.sh`, `SOC-Toolkit.ps1`) that, **read-only and without changing the system**, collect:

- Host identity and time (UTC), current and logged-on users, local administrators/sudoers.
- Failed and successful logins for the last *N* hours, with the top source IPs and accounts.
- Listening ports and established connections mapped to processes.
- Recently created or modified files in temp, startup and persistence locations.
- Firewall profile and rule summary.
- A timestamped text **and** CSV report plus a SHA-256 hash of the report.

Assessment criteria: correctness, read-only safety, error handling, clear report layout, documentation (usage, prerequisites, sample output). The course scripts `failed-login-report`, `port-monitor` and `log-analyser` (released Days 14–17) are your building blocks.

Recommended next steps after the course

- Free practice: TryHackMe/HackTheBox SOC-level rooms, Blue Team Labs Online, Splunk BOTS datasets.
- Certifications to consider: CompTIA Security+ / CySA+, Microsoft SC-200, Splunk Core User.

Path B — Network Analyst / Network Engineer

Network analysts keep traffic flowing and prove where it stops. Host-side command-line tools are how you separate "the network is down" from "this host is misconfigured".

Competencies and where they are taught

Competency	Linux	Windows	Days
IP addressing (IPv4/IPv6), private vs public	`ip -br addr`, `ip -6 addr`	`Get-NetIPAddress`, `Get-NetIPConfiguration`	11
Subnetting & CIDR	`ipcalc` (package), manual calculation	`Get-NetIPAddress PrefixLength`	11
Routing	`ip route`, `ip route get 8.8.8.8`	`Get-NetRoute`, `Find-NetRoute`	11, 27
DHCP	`journalctl -u systemd-networkd`, `/run/systemd/netif/leases`, `networkctl`	`ipconfig /all`, `Get-NetIPInterface -Dhcp`	11
DNS	`dig`, `host`, `resolvectl`	`Resolve-DnsName`, `Get-DnsClientCache`, `Clear-DnsClientCache`	13
ICMP, latency, loss, path	`ping`, `traceroute`, `tracepath`, `mtr`	`Test-Connection`, `tracert`, `pathping`	12
TCP/UDP and ports	`ss -tulpn`, `nc -zv`	`Test-NetConnection -Port`, `Get-NetTCPConnection`, `Get-NetUDPEndpoint`	14
Interfaces, MAC, ARP/ND	`ip link`, `ip neigh`	`Get-NetAdapter`, `Get-NetNeighbor`	15
Packet capture	`tcpdump -i any -w file.pcap`	`pktmon start --capture`, `pktmon etl2pcap`	15, 20
Wireshark analysis	Display filters, follow TCP stream, expert info	Same	15, 27
Nmap (authorised targets only)	`nmap -sT -p- localhost`, service detection	Nmap for Windows	20, 27
Network automation	Bash loops over host lists, CSV output	PowerShell loops, `Export-Csv`	22–25, 29

Structured troubleshooting model

Diagram — shown rendered on the website; text form:

```
flowchart TD
    A[Symptom reported] --> B{Link up?<br/>ip link / Get-NetAdapter}
    B -- no --> B1[Cable, Wi-Fi, driver, VLAN]
    B -- yes --> C{IP and gateway correct?<br/>ip addr, ip route / Get-NetIPConfiguration}
    C -- no --> C1[DHCP / static config]
    C -- yes --> D{Gateway reachable?<br/>ping gateway}
    D -- no --> D1[Local network / ARP]
    D -- yes --> E{Name resolves?<br/>dig / Resolve-DnsName}
```

```
    E -- no --> E1[DNS server, cache, records]
    E -- yes --> F{Port reachable?<br/>nc -zv / Test-NetConnection -Port}
    F -- no --> F1[Firewall, service down, routing]
    F -- yes --> G[Application layer: TLS, HTTP, auth]
```

Final project: Network Diagnostic Toolkit

Build `net-toolkit.sh` and `Net-Toolkit.ps1` that accept a **target host and port list** and produce a report containing:

- Local interfaces, addresses, gateway, DNS servers and routes.
- Gateway and target reachability (latency, loss) and path (traceroute/tracert).
- Forward and reverse DNS resolution, plus a comparison against a second resolver.
- TCP port tests for each requested port.
- Local listening ports and the owning processes.
- Optional: a 30-second capture filtered to the target, saved as `.pcap`. This needs authorisation and elevated rights.
- Output as text and CSV, with clear PASS/FAIL per layer.

Building blocks: `network-diagnostic`, `dns-check` and `port-monitor` (released Days 13–15).

Recommended next steps

- Certifications: CompTIA Network+, Cisco CCNA, Wireshark WCNA.
- Practice: GNS3/EVE-NG or Containerlab topologies, Wireshark sample captures (https://wiki.wireshark.org/SampleCaptures).

Path C — System Administrator / General IT Professional

System administrators keep servers and endpoints healthy, secure and up to date, and they automate everything they do twice.

Competencies and where they are taught

Competency	Linux	Windows	Days
User administration	`useradd`, `usermod`, `passwd`, `chage`	`New-LocalUser`, `Add-LocalGroupMember`, `Disable-LocalUser`	6
File management	`cp -a`, `rsync`, `find`	`Copy-Item`, `robocopy`	2–4
Permissions	`chmod`, `chown`, `umask`, ACLs (`setfacl`)	`Get-Acl`, `Set-Acl`, `icacls`	7
Storage monitoring	`df -h`, `du -sh`, `lsblk`, `findmnt`	`Get-Volume`, `Get-PSDrive`, `Get-Disk`	9
Service management	`systemctl status/restart/enable`, `journalctl -u`	`Get-Service`, `Restart-Service`, `Set-Service -StartupType`	8
Software installation	`apt`, `dpkg`, `snap`	`winget`, `Install-PSResource`	10
Backup checks	`tar`, `rsync --dry-run`, checksums	`robocopy /L`, `Get-FileHash`, `wbadmin get versions`	2, 9, 25
Log monitoring	`journalctl -p err`, `/var/log/syslog`	`Get-WinEvent -FilterHashtable @{LogName='System';Level=2}`	16
System maintenance	Updates, log rotation, cleanup	Windows Update status, temp cleanup	9, 10, 25
Remote administration	`ssh`, keys, `scp`, `sftp`	`Enter-PSSession`, `Invoke-Command`, SSH remoting	26
Automation	Bash scripts, cron, systemd timers	Advanced functions, Task Scheduler	21–25, 29

Daily, weekly and monthly routine (built during the course)

Frequency	Task	Tooling
Daily	Health report: CPU, memory, disk, failed services, errors in the last 24h	`system-health`, `daily-report` scripts (Days 25, 29)
Daily	Backup job success check	Exit codes + log parsing (Days 5, 16)
Weekly	Patch status, pending reboots	`apt list --upgradable`, `winget upgrade` (Day 10)

Frequency	Task	Tooling
Weekly	Disk growth trend	`disk-monitor` script (Day 9)
Monthly	Account review: inactive users, admin group membership	Day 6 + Day 19 audits
Monthly	Restore test from backup	Lab practice

Final project: System Health Monitoring Toolkit

Build `health-toolkit.sh` and `Health-Toolkit.ps1` that:

- Collect CPU load, memory usage, disk usage per volume (with warning/critical thresholds as parameters), uptime and pending updates.
- List failed/stopped-but-automatic services and the top 5 processes by CPU and memory.
- Summarise errors from the system log in the last 24 hours.
- Check that the last backup file or log is newer than *N* hours.
- Write a dated text and CSV report, keep 14 days of reports, and exit with a non-zero code when any check is critical, so a scheduler or monitoring system can alert.
- Include installation instructions for cron/systemd timers and Task Scheduler.

Building blocks: `system-info` (Day 1), `process-monitor` (Day 8), `disk-monitor` (Day 9), `system-health` (Day 25) and `daily-report` (Day 29).

Recommended next steps

- Certifications: CompTIA Linux+ / A+, LPIC-1, Red Hat RHCSA, Microsoft AZ-800/AZ-104.
- Practice: build a home lab with two Ubuntu VMs and one Windows VM, and automate their patching and health reporting.