Ubuntu Academy: Using Ubuntu Desktop, 2026

Essential Desktop Administration, Productivity, and Customization on Ubuntu

Onno W. Purbo
onno@indo.net.id
onno@itts.ac.id
Twitter onnowpurbo

Institut Teknologi Tangerang Selatan (ITTS)
OnnoCenter
2026

Copyright

Copyright © 2026
This book is licensed under the **Creative Commons Attribution 4.0 International License (CC BY 4.0)**. You are free to share and adapt this material for any purpose, including commercial use, provided that appropriate credit is given, a link to the license is included, and any changes made are indicated.
License: https://creativecommons.org/licenses/by/4.0/

Preface

Alhamdulillāh, so that this book, **Ubuntu Academy: Using Ubuntu Desktop, 2026**, can be completed. We are also deeply thankful to everyone who has contributed, supported, and inspired the preparation of this manuscript, including educators, practitioners, and the open-source community.

This book is written to help readers understand and use **Ubuntu Desktop** in a practical, friendly, and confident way. It is intended for students, teachers, professionals, and general users who want to learn Ubuntu not only as an operating system, but also as a platform for productivity, exploration, and continuous learning.

We hope this book can become a useful companion for study, teaching, and daily practice, and encourage more readers to grow with the spirit of *open-source* collaboration and learning.

May this book be beneficial and useful for all.

Jakarta, March 2026

Table Of Content

Copyright **2**
Preface **3**
Table Of Content **4**
Front Matter **28**
How to Use This Book 28
Recommended way to read and practice 28
How different readers can use this book 29
Study strategy for best results 29
Illustration 29
Suggested reading flow 29
Screenshot notes 30
Lab Environment and Practice Setup 31
Recommended practice environments 31
Option 1 — Virtual Machine 31
Option 2 — Live USB Session 32
Option 3 — Full Install on Physical Machine 32
Option 4 — Dual Boot 32
Minimum practice checklist 32
Suggested lab folder structure 33
Practice account recommendation 33
What to practice repeatedly 34
Role of screenshots in the lab 34
Illustration 34
Screenshot notes 35
Conventions Used in This Book 36
Typographic conventions 36
Command-line conventions 36
Path conventions 37
GUI path conventions 37
Warning and tip conventions 37
Output variability 38
Naming conventions in labs 38
Screenshot code conventions 39
Illustration 39
Learning Outcomes 39
General learning outcome 40
Specific learning outcomes 40
1. Understand Ubuntu Desktop as a platform 40
2. Navigate the Ubuntu Desktop environment 40
3. Prepare and install Ubuntu Desktop 40
4. Manage applications and software sources 40

5. Use Ubuntu Desktop productively in daily work....41
6. Customize and optimize the desktop....41
7. Manage users and permissions....41
8. Configure hardware, devices, and multimedia....41
9. Configure networking for desktop users....41
10. Secure and maintain Ubuntu Desktop....41
11. Troubleshoot and recover from problems....42
12. Participate in the Ubuntu ecosystem....42
Performance-oriented outcome....42
Illustration....42
Ubuntu Desktop Competency Map....42
Competency philosophy....43
Domain-based competency map....43
Domain 1 — Ubuntu Desktop Foundations....43
Domain 2 — Desktop Navigation and Daily Use....43
Domain 3 — Installation and First-Time Setup....44
Domain 4 — Software Management....44
Domain 5 — Customization and User Experience....44
Domain 6 — Users, Permissions, and Local Administration....45
Domain 7 — Devices, Multimedia, and Peripherals....45
Domain 8 — Networking for Desktop Users....45
Domain 9 — Security, Privacy, and Maintenance....46
Domain 10 — Troubleshooting and Recovery....46
Domain 11 — Community and Continuing Growth....46
Competency progression table....47
How instructors can use this map....47
Illustration....47
Part I — Foundations of Ubuntu Desktop....48
Chapter 1 — Understanding Ubuntu Desktop....48
Ubuntu Desktop vs Ubuntu Server....49
Ubuntu Release Model: LTS vs Interim Releases....49
Official Ubuntu Flavors and Desktop Choices....50
Why Ubuntu Desktop for Learning, Work, and Daily Use....51
Community, Documentation, and Support Channels....52
Chapter Reflection....52
Key Takeaways....53
Chapter 2 — Linux Desktop Technology in Context....54
What a Linux Desktop Environment Is....54
GNOME as the Default Ubuntu Desktop....55
Wayland vs X.Org in Ubuntu Desktop....56
Core Desktop Components....57
Display server....57
Window manager....57
File manager....57

Settings framework..... 57
Widget and toolkit basics..... 58
Workspaces, Notifications, and Modern Desktop Workflows.....58
Choosing the Right Desktop for Performance, Usability, and Hardware Constraints. 59
Chapter Reflection..... 60
Key Takeaways.....60
Chapter 3 — Touring the Ubuntu Desktop.....61
First Login Experience..... 61
Top Bar, Dock, Activities Overview, and App Grid..... 62
Screenshots:..... 63
Files (Nautilus) and Folder Navigation.....64
CLI example: open the Home folder path in Terminal.....65
CLI example: list common folders in the Home directory.....65
CLI example: create a practice folder..... 65
Search, Notifications, Calendar, and Quick Settings..... 66
CLI example: verify current date and time..... 67
CLI example: check hostname and current user..... 67
Window Management and Workspaces..... 68
CLI example: open several applications for workspace practice..... 68
CLI example: inspect running graphical applications.....68
Accessibility Features..... 69
Power, Session, Lock, Logout, Suspend, Restart, and Shutdown.....70
CLI example: lock the current session.....71
CLI example: reboot or power off from Terminal.....71
Keyboard Shortcuts for Productive Daily Use.....72
CLI example: launch Terminal quickly and inspect the desktop session..... 72
CLI example: check current graphical session type.....73
Guided Desktop Walkthrough Activity.....73
Chapter Reflection..... 74
Key Takeaways.....74
Part II — Installation and First-Time Setup..... 75
Chapter 4 — Preparing and Installing Ubuntu Desktop.....75
Hardware Requirements and Compatibility Checks..... 75
CLI example: inspect hardware from a live session.....76
CLI example: detect hardware devices.....76
Choosing the Right Ubuntu Desktop Edition..... 77
Creating Bootable Installation Media..... 78
Rufus.....79
Startup Disk Creator..... 79
dd.....79
CLI example: identify the USB device before writing..... 80
CLI example: unmount the USB before writing.....80
CLI example: write the Ubuntu ISO to USB with dd.....80
CLI example: flush remaining writes..... 80

Trying Ubuntu in Live Mode....81
CLI example: confirm the session type in live mode....81
CLI example: check network connection....81
Clean Install, Dual Boot, and VM Installation....82
CLI example: check current partitions before planning a dual boot....83
Installer Options....83
Minimal vs full install....84
Third-party drivers and codecs....84
Disk partitioning basics....84
Encryption options....85
First Boot and Initial Configuration....85
CLI example: refresh package metadata after first boot....86
CLI example: install available updates....86
CLI example: check which desktop session is running....86
CLI example: see which packages can still be upgraded....86
Guided Installation Practice....87
Chapter Reflection....88
Key Takeaways....88
Chapter 5 — Post-Installation Essentials....89
Connecting to the Internet....89
CLI example: verify network interface status....89
CLI example: test internet access....90
Updating the System After Install....91
CLI example: refresh package metadata....91
CLI example: install available package updates....91
CLI example: list packages that can still be upgraded....92
Installing Proprietary Drivers....93
CLI example: identify graphics hardware....93
CLI example: ask Ubuntu which drivers are available....93
NVIDIA....94
CLI example: install recommended NVIDIA driver....95
CLI example: verify loaded NVIDIA driver....95
AMD....96
CLI example: identify AMD graphics adapter....96
CLI example: inspect renderer information....97
Broadcom and Realtek Wi-Fi....98
CLI example: identify wireless hardware....98
CLI example: see whether a wireless interface exists....98
Language Packs, Timezone, and Input Methods....99
CLI example: display current locale....100
CLI example: display time and timezone....100
CLI example: change timezone....100
Enabling Extra Multimedia Support....101
CLI example: install extra multimedia support....102

CLI example: install a multimedia player 102
Essential First Applications to Install 103
CLI example: install some useful starter applications 103
CLI example: install a Snap application 103
CLI example: enable Flatpak support 103
Making Ubuntu Ready for Daily Desktop Use 104
CLI example: quick readiness checklist 104
Guided Post-Install Activity 106
Chapter Reflection 106
Key Takeaways 106
Part III — Applications and Software Management 107
Chapter 6 — Managing Applications on Ubuntu Desktop 107
Understanding Ubuntu Software Sources 107
GUI App Installation with App Center / Ubuntu Software 108
APT Fundamentals 109
apt update 109
apt upgrade 110
apt install 110
apt remove 111
apt search 111
Working with .deb Packages 112
dpkg 112
CLI example: install a local .deb package 112
CLI example: remove a package installed with dpkg 113
CLI example: list installed packages with dpkg 113
GUI helpers 113
Snap Packages 113
Search 114
Install 114
Refresh 114
Remove 114
List 114
Confinement concepts 115
Flatpak and Flathub 116
Installation 116
Integration 116
Updating apps 117
PPAs and Third-Party Repositories Safely 118
CLI example: add a PPA 118
CLI example: remove a broken or unwanted PPA source file 118
Cleaning Unused Packages and Managing Disk Space 119
CLI example: remove unneeded dependency packages 119
CLI example: check disk usage 119
CLI example: view installed Snaps 119

Guided Application Management Lab....120
Chapter Reflection....121
Key Takeaways....121
Chapter 7 — Everyday Ubuntu Desktop Applications....122
Web Browsers and Safe Browsing....122
CLI example: install Firefox from Ubuntu repositories....123
CLI example: search for Chromium....123
CLI example: launch the browser from Terminal....123
Email and Communication Apps....125
CLI example: search for a desktop email client....125
CLI example: install Thunderbird....125
CLI example: launch the email client....126
Office and Document Productivity....127
CLI example: search for LibreOffice....127
CLI example: install LibreOffice....127
CLI example: start Writer....128
Media Playback: Music, Video, and Podcasts....129
CLI example: install VLC with APT....129
CLI example: install VLC as a Snap....129
CLI example: launch Rhythmbox....130
CLI example: launch VLC....130
Image Editing and Creative Tools....131
CLI example: install GIMP....131
CLI example: install Inkscape....131
CLI example: launch GIMP....132
PDF, eBook, and Note-Taking Apps....133
CLI example: search for a note-taking tool....133
CLI example: install a PDF-annotation or note-taking application....133
CLI example: launch a note-taking app....133
Social, Messaging, and Collaboration Apps....135
CLI example: download the Zoom installer....135
CLI example: install Zoom....135
CLI example: launch Zoom....136
App Selection Strategy: Native Package vs Snap vs Flatpak vs .deb....138
CLI example: compare the same idea across package methods....139
Comparing apt, snap, flatpak and .deb....140
Guided Everyday Apps Activity....140
Chapter Reflection....141
Key Takeaways....141
Chapter 8 — Running Windows-Oriented Workflows on Ubuntu....142
Replacing Common Windows Applications....142
CLI example: install common Windows-style equivalents....143
CLI example: search for a browser alternative....143
File Compatibility and Cross-Platform Formats....145

CLI example: inspect a downloaded cross-platform document..... 145
CLI example: demonstrate Linux path style.....145
CLI example: show case sensitivity..... 146
Using Wine and Compatibility Layers..... 147
CLI example: install Wine from Ubuntu repositories.....147
CLI example: perform Wine’s first configuration..... 147
CLI example: inspect Wine’s working folder..... 148
Running Selected Windows Programs..... 149
CLI example: run a Windows installer in Wine.....149
CLI example: list Wine-installed program files.....149
CLI example: launch an installed Windows application..... 150
Virtual Machines for Legacy Software..... 151
CLI example: check whether hardware virtualization support is visible..... 151
CLI example: install VirtualBox on Ubuntu.....151
CLI example: list registered virtual machines..... 152
Migration Tips for Users Coming from Windows/macOS.....153
CLI example: show the current home directory..... 154
CLI example: demonstrate file path separators..... 154
CLI example: create a migration practice folder..... 154
Guided Migration Activity..... 154
Chapter Reflection..... 155
Key Takeaways.....155
Part IV — Desktop Customization and User Experience..... 156
Chapter 9 — Customizing the GNOME Desktop.....156
Appearance Settings..... 156
Themes.....158
CLI example: inspect current GTK theme.....158
CLI example: change the GTK theme.....158
Icons..... 160
CLI example: inspect the current icon theme.....160
CLI example: set a different icon theme..... 160
Wallpapers.....162
CLI example: set a wallpaper with GSettings..... 162
CLI example: inspect the current wallpaper.....162
Dark Mode..... 164
CLI example: prefer dark mode..... 164
CLI example: inspect current color-scheme preference..... 164
Dock and Panel Customization.....165
Fonts, Cursor, and Scaling..... 166
CLI example: inspect interface font.....166
CLI example: inspect cursor theme..... 166
CLI example: check current scaling factor.....166
GNOME Extensions.....168
CLI example: list enabled GNOME extensions.....168

CLI example: list all installed GNOME extensions....168
GNOME Tweaks....170
CLI example: install GNOME Tweaks....170
CLI example: launch GNOME Tweaks....170
Advanced Tweaks with GSettings and dconf....172
CLI example: read a GNOME preference with GSettings....172
CLI example: enable weekday display in the top bar clock....172
CLI example: inspect a dconf path....172
Desktop Icons, Startup Apps, and Session Behavior....174
CLI example: inspect autostart entries....174
CLI example: disable a startup application by moving its autostart file....174
CLI example: check the current desktop session....175
Designing a Clean, Productive Desktop Workspace....176
Guided Customization Activity....177
Chapter Reflection....177
Key Takeaways....177
Chapter 10 — Files, Search, and Daily Workspace Management....178
Using Files (Nautilus) Efficiently....178
CLI example: open the current folder in Terminal....179
CLI example: list the contents of the Home directory....179
CLI example: launch Nautilus from Terminal....179
Bookmarks, Tabs, and File Operations....181
CLI example: create a structured workspace folder tree....181
CLI example: copy and move files....181
Search and Recent Files....183
CLI example: fast search with plocate....183
CLI example: case-insensitive search....183
CLI example: live search with find....184
CLI example: search with detailed listing....184
Archives and Compression....185
CLI example: create a ZIP archive....185
CLI example: create a tar.gz archive....185
CLI example: extract a ZIP archive....186
External Drives and File Transfers....187
CLI example: identify attached storage devices....187
CLI example: copy a file to an external drive....187
CLI example: safely unmount a removable drive....188
Trash, Recovery, and Safe Deletion....189
CLI example: move a file to Trash using the command line....189
CLI example: inspect the user Trash location....189
CLI example: permanently remove a file....189
CLI example: confirm a file is gone....190
Screenshots, Screen Recording, and Sharing Files....191
CLI example: make sure the Pictures/Screenshots folder exists....191

CLI example: inspect recently created screenshots..... 191
CLI example: share a file by copying it to a common exchange folder.....191
Organizing a Practical Desktop Workflow.....193
CLI example: create a simple daily workspace structure.....193
CLI example: move finished work into an archive folder.....193
CLI example: search the workspace for PDFs only.....194
Guided File Management Activity..... 195
Chapter Reflection..... 195
Key Takeaways.....195
Part V — Users, Permissions, and Local Administration..... 196
Chapter 11 - Managing Users.....196
Creating and Managing Users..... 196
CLI example: create a new user..... 197
CLI example: list the user’s home directory.....197
CLI example: modify a user account.....198
CLI example: remove a user account..... 198
Password Changes and Account Recovery Basics.....199
CLI example: change the current user’s password.....199
CLI example: administrator changes another user’s password..... 199
CLI example: check who you are before resetting anything..... 200
User Switching and Multi-User Desktops..... 201
CLI example: show the current logged-in user..... 201
CLI example: list currently logged-in sessions.....201
CLI example: list session information through loginctl.....201
Groups and Administrative Privileges.....203
CLI example: show a user’s groups.....203
CLI example: add a user to the sudo group.....203
CLI example: add a user to several desktop-related groups..... 204
sudo Basics in Desktop Administration.....205
CLI example: update the system with sudo..... 205
CLI example: what happens without sudo on a restricted action.....205
CLI example: validate or refresh sudo credentials.....206
CLI example: list a user’s sudo privileges.....206
File and Folder Permissions for Desktop Users..... 207
CLI example: inspect file permissions.....207
CLI example: make a script executable.....207
CLI example: change file ownership.....208
Ownership, Sharing, and Collaborative Folders..... 209
CLI example: create a shared project folder.....209
CLI example: grant another user ACL permissions..... 210
CLI example: access a Samba share from the desktop side.....210
Guest-style Usage, Privacy, and Session Hygiene.....211
CLI example: lock the current session..... 211
CLI example: check the current desktop session identity..... 211

CLI example: see what autostarts or personal files might remain visible..... 212
Guided User and Permissions Lab..... 213
Chapter Reflection..... 213
Key Takeaways..... 213
Chapter 12 — Desktop Administration Tools..... 214
Settings App vs Command-Line Administration..... 214
CLI example: identify the desktop session before troubleshooting..... 215
CLI example: open the graphical Settings app from Terminal..... 215
Privilege Escalation Safely..... 217
CLI example: run an administrative command properly..... 217
CLI example: see what happens without sudo..... 217
CLI example: refresh sudo credentials without running another command..... 218
Logs, Status, and Basic Diagnostic Commands..... 219
CLI example: check basic system identity and uptime..... 220
CLI example: inspect storage and memory quickly..... 220
CLI example: view recent system log entries..... 220
CLI example: show only current-boot logs..... 220
Monitoring Processes and System Activity..... 222
CLI example: list processes by memory size..... 222
CLI example: live monitoring with top..... 223
CLI example: kill a stuck process..... 223
CLI example: graphical process monitor..... 223
Managing Startup Items and Sessions..... 224
CLI example: open Startup Applications graphically..... 224
CLI example: inspect user autostart entries..... 224
CLI example: create an autostart directory if needed..... 225
CLI example: inspect active sessions..... 225
When to Use GUI Tools and When to Use Terminal..... 226
CLI example: compare a GUI task and a Terminal task..... 227
Desktop Support Best Practices for Labs, Schools, and Shared PCs..... 228
CLI example: quick shared-PC health check..... 228
CLI example: identify current startup items on a shared machine..... 229
CLI example: launch System Monitor for a support session..... 229
Guided Desktop Administration Lab..... 230
Chapter Reflection..... 230
Key Takeaways..... 231
Part VI — Hardware, Devices, and Multimedia..... 232
Chapter 13 — Hardware Detection and Driver Management..... 232
Understanding Ubuntu Hardware Support..... 232
Detecting Hardware..... 233
lspci..... 233
CLI example: list major PCI devices..... 233
CLI example: get more detailed PCI information..... 233
lsusb..... 234

CLI example: list USB devices....234
lshw....234
CLI example: summarize hardware....234
inxi....235
CLI example: get a readable system summary....235
Graphics Hardware and GPU Drivers....237
CLI example: detect graphics hardware....237
CLI example: ask Ubuntu what drivers are available....237
CLI example: install recommended drivers automatically....238
CLI example: verify the NVIDIA driver....238
Hybrid Graphics and PRIME Concepts....240
CLI example: see available PRIME profiles....240
CLI example: switch to on-demand mode....240
CLI example: switch to NVIDIA-only mode....241
Audio Devices and Troubleshooting Basics....242
CLI example: test basic sound playback....242
CLI example: open ALSA mixer....242
CLI example: list audio devices....243
CLI example: reset user PulseAudio settings....243
Bluetooth Devices and Pairing....244
CLI example: check for USB Bluetooth hardware....244
CLI example: show Bluetooth service status....244
CLI example: open Bluetooth control in the desktop....244
Power and Battery Management for Laptops....246
CLI example: inspect power devices....246
CLI example: show detailed battery information....246
CLI example: install TLP....247
CLI example: check TLP status....247
External Displays, Docking, and Multi-Monitor Setups....248
CLI example: list display outputs with xrandr....248
CLI example: enable an external monitor to the right....248
CLI example: mirror the laptop screen....248
Screenshot notes:....249
Guided Hardware and Driver Lab....250
Chapter Reflection....250
Key Takeaways....251
Chapter 14 — Printers, Scanners, Storage, and Removable Media....252
Printer Setup with CUPS....252
CLI example: verify that CUPS is installed....253
CLI example: check CUPS service status....253
CLI example: install graphical printer tools if needed....253
Adding and Managing Printers....255
CLI example: show the status of all printers....255
CLI example: print a test file....255

CLI example: cancel a print job....256
CLI example: open the printer management window....256
Scanners and Document Capture....257
CLI example: inspect USB devices for scanner or multifunction hardware....257
CLI example: follow kernel messages while plugging in a scanner....257
CLI example: inspect recent kernel messages....258
USB Devices and Automount Behavior....259
CLI example: inspect block devices before and after inserting a USB drive....259
CLI example: list all currently mounted filesystems....259
CLI example: follow log messages when inserting a USB drive....260
Optical Media and Removable Drives....261
CLI example: inspect mounted optical media....261
CLI example: check optical drive capabilities....261
Filesystems and Mount Points in Desktop Context....262
CLI example: show filesystems and mount points....262
CLI example: mount a filesystem manually....262
CLI example: unmount it safely....263
Troubleshooting Peripherals and Device Detection....264
CLI example: inspect printer-related status....264
CLI example: inspect block devices....264
CLI example: inspect USB peripherals....265
CLI example: inspect recent hardware messages....265
CLI example: monitor new events live....265
Guided Peripheral Management Lab....266
Chapter Reflection....266
Key Takeaways....267
Part VII — Networking on Ubuntu Desktop....268
Chapter 15 — Network Configuration for Desktop Users....268
NetworkManager Basics....268
CLI example: list network devices with nmcli....269
CLI example: show saved connection profiles....269
Wi-Fi and Ethernet Setup....270
CLI example: scan available Wi-Fi networks....270
CLI example: connect to a Wi-Fi network....270
CLI example: inspect IP address state on Ethernet and Wi-Fi....270
IP Addressing Basics....272
CLI example: show addresses assigned to interfaces....272
CLI example: show routing information....272
Static IP, DNS, and Proxy Configuration....274
CLI example: set a static IPv4 address with nmcli....274
CLI example: inspect current DNS configuration source....274
CLI example: verify DNS servers manually....275
VPN and Mobile Broadband....276
CLI example: list available NetworkManager connection types....276

CLI example: bring up a saved VPN profile....276
CLI example: display current device status after VPN activation....277
Hostname Configuration....278
CLI example: show the current hostname....278
CLI example: set a new hostname....278
Connection Profiles and Troubleshooting....280
CLI example: inspect a specific connection profile....280
CLI example: reconnect a profile....280
CLI example: check route to the gateway and beyond....280
CLI example: test hostname resolution separately....281
Using ping, ip, and nmcli for Quick Diagnostics....282
CLI example: fast all-in-one network check....282
Guided Networking Lab....284
Chapter Reflection....284
Key Takeaways....285
Chapter 16 — Desktop Sharing and Connectivity....286
Bluetooth Sharing and Local Transfers....286
CLI example: verify that Ubuntu sees a Bluetooth adapter....287
CLI example: check Bluetooth service status....287
CLI example: open the Bluetooth settings page....287
Network File Access Basics....288
CLI example: open a Samba location from the command line....288
CLI example: check whether Samba client tools are available....288
CLI example: list shares available on a Samba server....289
CLI example: connect to a share interactively....289
Remote Desktop and Screen Sharing....290
CLI example: identify the current hostname before sharing remotely....290
CLI example: open the sharing-related desktop settings....291
CLI example: verify local address before remote access....291
Secure File Transfer Basics....292
CLI example: copy a file securely with scp....292
CLI example: copy a directory recursively....292
CLI example: open an SFTP session....292
Internet Sharing / Hotspot Setup....294
CLI example: review Wi-Fi device capability and state....294
CLI example: create a Wi-Fi hotspot with NetworkManager....294
CLI example: inspect the created hotspot connection....294
Working on Home, Campus, and Office Networks....296
CLI example: inspect connection information quickly....296
CLI example: show current proxy-related environment variables....297
Practical Connectivity Scenarios for Daily Desktop Use....298
Scenario 1 — Connect to campus Wi-Fi and verify access....298
CLI example....298
Scenario 2 — Open a shared folder from a Windows or Mac machine....298

Scenario 3 — Send a file securely to a colleague on the same LAN.....298
CLI example.....298
Scenario 4 — Turn a laptop into a temporary hotspot..... 298
Scenario 5 — Diagnose “internet works by IP but not by name”..... 299
Guided Sharing and Connectivity Lab..... 300
Chapter Reflection..... 300
Key Takeaways.....300
Part VIII — Security, Privacy, and Reliability.....301
Chapter 17 — Securing Ubuntu Desktop..... 301
Strong Passwords and Account Protection..... 301
CLI example: change the current user password..... 302
CLI example: inspect password-aging settings for a user..... 302
Screen Locking and Session Security..... 304
CLI example: lock the current session.....304
CLI example: show the current desktop session type.....304
Full-Disk Encryption Basics..... 306
CLI example: view block devices and encryption-related layout.....306
CLI example: confirm the root device in use.....306
Firewall with UFW.....308
CLI example: check firewall status.....308
CLI example: allow SSH only from the local subnet..... 308
CLI example: enable the firewall.....308
CLI example: check the active rules..... 309
AppArmor Basics on Ubuntu..... 310
CLI example: check AppArmor status.....310
CLI example: inspect AppArmor package presence.....310
CLI example: view active AppArmor-related lines..... 310
Trusted Repositories and Safe Software Installation.....312
CLI example: refresh repository metadata.....312
CLI example: add a PPA carefully..... 312
CLI example: remove an untrusted or outdated source file..... 312
Snap/Flatpak Sandboxing Benefits.....314
CLI example: list installed snaps.....314
CLI example: install a Snap application.....314
CLI example: verify Flatpak support..... 314
Browser Security, Downloads, and Phishing Awareness.....316
CLI example: verify that the browser package is up to date through APT metadata 316
CLI example: inspect saved passwords and keys tool availability.....316
Intro to Malware Scanning and Safe Habits..... 318
CLI example: install ClamAV.....318
CLI example: update or refresh signatures.....318
CLI example: scan the home directory..... 318
CLI example: install a rootkit checker..... 319

Backup Encryption and Recovery Planning....320
CLI example: install Déjà Dup if needed....320
CLI example: install duplicity for encrypted backup workflows....321
CLI example: launch the GNOME password/key manager....321
Guided Desktop Security Lab....322
Chapter Reflection....322
Key Takeaways....323
Chapter 18 — Updates, Maintenance, and Backup....324
Manual and Automatic Updates....324
CLI example: refresh package metadata....325
CLI example: apply package upgrades....325
CLI example: list upgradeable packages....325
Security Updates and Patch Strategy....327
CLI example: inspect the current Ubuntu release-upgrade prompt policy....327
CLI example: upgrade to a newer release when appropriate....327
unattended-upgrades Basics....329
CLI example: install unattended-upgrades....329
CLI example: check whether the service is present....329
CLI example: review configuration file presence....329
Routine System Maintenance....331
CLI example: check free disk space....331
CLI example: inspect system uptime....331
CLI example: list installed packages count roughly....331
Cleaning Cache and Reducing Bloat....333
CLI example: remove automatically installed unused packages....333
CLI example: clean APT package cache....333
CLI example: inspect APT cache size before or after cleanup....333
Backup with Déjà Dup....335
CLI example: install Déjà Dup....335
CLI example: launch Déjà Dup....335
CLI example: prepare a simple backup target folder....335
Snapshot Concepts with Timeshift....337
CLI example: install Timeshift....337
CLI example: launch Timeshift....337
Restore Workflows and Recovery Mindset....339
CLI example: create a safe restore target....339
CLI example: verify recovered files after restore....339
Screenshot notes:....339
Guided Maintenance and Backup Lab....340
Chapter Reflection....340
Key Takeaways....340
Part IX — Troubleshooting Ubuntu Desktop....341
Chapter 19 — Solving Common Desktop Problems....341
Frozen Applications and Unresponsive Sessions....341

CLI example: inspect heavy processes..... 342
CLI example: live monitoring with top..... 342
CLI example: terminate a stuck application..... 342
CLI example: open the graphical System Monitor..... 342
Software Installation Problems..... 344
CLI example: refresh package lists.....344
CLI example: search for a package..... 344
CLI example: diagnose a package not found situation..... 345
CLI example: repair an interrupted installation state.....345
CLI example: reconfigure a misbehaving package.....345
Broken Dependencies and Repository Issues.....346
CLI example: repair broken dependencies..... 346
CLI example: remove unneeded packages..... 346
CLI example: inspect repository source files..... 347
CLI example: remove an unwanted repository source and refresh..... 347
Slow Startup and Performance Bottlenecks..... 348
CLI example: inspect uptime and load.....348
CLI example: check memory and disk pressure.....348
CLI example: inspect startup applications..... 348
CLI example: open Startup Applications graphically.....349
Sound, Wi-Fi, Bluetooth, and Display Problems.....350
CLI example: test sound output..... 350
CLI example: open ALSA mixer.....350
CLI example: reset PulseAudio settings..... 350
CLI example: inspect Bluetooth adapter presence..... 350
CLI example: inspect network connection state.....351
CLI example: inspect current display outputs..... 351
Login Issues and User Session Problems..... 352
CLI example: confirm the current user after successful login..... 352
CLI example: list active sessions.....352
CLI example: inspect user autostart items that may destabilize the session.....352
Graphics Driver Conflicts..... 354
CLI example: identify graphics hardware.....354
CLI example: inspect available drivers..... 354
CLI example: inspect current PRIME profile on a hybrid laptop..... 355
CLI example: verify NVIDIA driver health..... 355
App Crashes and Extension Conflicts..... 356
CLI example: list enabled GNOME extensions.....356
CLI example: disable one extension for testing..... 356
CLI example: inspect all installed extensions..... 356
CLI example: reconfigure a recently broken package.....357
Guided Desktop Troubleshooting Lab..... 358
Chapter Reflection..... 358
Key Takeaways.....359

Chapter 20 — Boot and Recovery Troubleshooting....360
Understanding the Boot Process at a Desktop Level....360
GRUB Basics....362
CLI example: inspect the current boot entry source file....362
CLI example: show installed kernel images....362
CLI example: show initramfs images....362
CLI example: update GRUB configuration after a repair....363
Common Boot Failures....364
CLI example: inspect the current kernel version on a working system....364
CLI example: view recent current-boot logs on a system that did boot....364
initramfs and Recovery Mode Basics....366
CLI example: rebuild the initramfs for the current kernel....366
CLI example: rebuild initramfs for all installed kernels....366
CLI example: confirm the initramfs files exist....366
Screenshot notes:....366
Boot-Repair Tool....368
CLI example: install Boot-Repair in a live environment....368
Screenshot notes:....368
Live USB Rescue Workflows....370
CLI example: identify disks and partitions from a live session....370
CLI example: mount the installed system partition in the live session....370
CLI example: launch the Disks utility from the live session....370
Screenshot notes:....371
Saving User Files Before Repair....372
CLI example: copy a user's Documents folder to an external drive....372
CLI example: verify that the copied files exist....372
CLI example: copy user data to a second internal or network target....372
Practical Desktop Recovery Checklist....374
CLI example: quick recovery toolkit summary from a live session....374
Guided Boot Recovery Lab....376
Chapter Reflection....376
Key Takeaways....376
Part X — Open Source Participation and Ubuntu Ecosystem....377
Chapter 21 — Ubuntu Community and Open Source Engagement....377
Ubuntu Governance and Community Structure....377
Free Software and Open Source Principles....379
Screenshot notes:....379
Common Licenses: GPL, MIT, Apache....381
CLI example: inspect license files commonly installed on Ubuntu....381
CLI example: read the beginning of the GPL text....382
Finding Help....383
Official documentation....383
CLI example: use built-in command help....383
CLI example: use a man page....383

Ask Ubuntu..... 384
Ubuntu Discourse..... 384
Forums and chat communities.....384
Reporting Bugs and Giving Useful Feedback.....385
CLI example: gather Ubuntu version info for a bug report.....385
CLI example: gather kernel version info..... 385
CLI example: capture recent session logs.....385
Contributing Through Testing, Documentation, Translation, and Advocacy..... 387
CLI example: inspect current locale before thinking about translation-related context..... 387
CLI example: gather system information when testing a bug or feature.....387
Building Good Habits as an Ubuntu Desktop User.....389
CLI example: use built-in help for a command.....389
CLI example: save troubleshooting notes for yourself.....389
Guided Community and Contribution Lab.....391
Chapter Reflection.....391
Key Takeaways.....392
Part XI — Productivity, Practice, and Real-World Use..... 393
Chapter 22 — Everyday Desktop Best Practices..... 393
Productive Workflows with Workspaces and Shortcuts.....393
CLI example: confirm the current desktop environment..... 394
CLI example: launch Terminal quickly for a desktop workflow.....394
CLI example: inspect currently running graphical apps..... 394
Screenshot notes:.....395
File Discipline and Backup Habits..... 395
CLI example: create a structured workspace folder tree..... 395
CLI example: move project work into an active workspace..... 395
CLI example: prepare a safe backup location.....396
Screenshot notes:.....396
Safe Update Habits.....396
CLI example: refresh package metadata.....396
CLI example: install upgrades safely..... 397
CLI example: inspect upgradeable packages first..... 397
Screenshot notes:.....397
Resource-Aware Desktop Usage on Low-End PCs..... 398
CLI example: inspect memory usage.....398
CLI example: inspect disk and workload.....398
CLI example: inspect startup entries.....398
Best Practices for Students, Teachers, Office Users, and Home Users..... 400
Students.....400
Teachers..... 400
Office users.....400
Home users.....401
CLI example: create role-based folders.....401

Maintaining a Stable Yet Flexible Ubuntu Desktop....402
CLI example: inspect enabled GNOME extensions....402
CLI example: inspect user startup items....402
CLI example: keep notes on non-default changes....402
Guided Everyday Best Practices Lab....404
Chapter Reflection....404
Key Takeaways....404
Chapter 23 — Guided Hands-On Labs....405
How to Use These Labs....405
Lab 1: Install Ubuntu Desktop in a VM....406
Goal....406
Skills covered....406
Why this lab matters....406
Preparation....406
Tasks....406
Verification....407
Reflection....407
Lab 2: Configure a Full Daily-Use Workstation....408
Goal....408
Skills covered....408
Why this lab matters....408
Preparation....408
Tasks....408
Verification....408
Reflection....409
Lab 3: Install Apps via APT, Snap, Flatpak, and .deb....410
Goal....410
Skills covered....410
Why this lab matters....410
Tasks....410
Verification....411
Reflection....411
Lab 4: Customize GNOME for Productivity....412
Goal....412
Skills covered....412
Why this lab matters....412
Tasks....412
Verification....412
Reflection....413
Lab 5: Add Printer, Scanner, Bluetooth Headset, and Second Monitor....414
Goal....414
Skills covered....414
Why this lab matters....414
Tasks....414

Verification 415
Reflection 415
Lab 6: Configure Wi-Fi, Static IP, Firewall, and VPN 416
Goal 416
Skills covered 416
Why this lab matters 416
Tasks 416
Verification 416
Reflection 417
Lab 7: Create Users and Secure the Desktop 418
Goal 418
Skills covered 418
Why this lab matters 418
Tasks 418
Verification 419
Reflection 419
Lab 8: Backup, Restore, and Recover from Common Failures 420
Goal 420
Skills covered 420
Why this lab matters 420
Tasks 420
Verification 420
Reflection 421
Lab 9: Fix a Boot Issue Using Recovery Tools 422
Goal 422
Skills covered 422
Why this lab matters 422
Tasks 422
Verification 423
Reflection 423
Lab 10: Build Your Own Ubuntu Desktop Checklist 424
Goal 424
Skills covered 424
Why this lab matters 424
Tasks 424
CLI example: create a checklist folder and file 425
Example checklist starter 425
Verification 425
Reflection 425
Chapter Reflection 426
Key Takeaways 426
Appendix A — Ubuntu Desktop Quick Reference 427
1. Essential Commands for APT, Snap, and Flatpak 427
APT quick commands 427

.deb quick commands....428
Snap quick commands....428
Flatpak quick commands....429
2. Networking Quick Commands....430
Static IP / Wi-Fi / hotspot quick examples....430
Firewall quick commands....431
3. Hardware Information Commands....431
Graphics / driver quick commands....431
4. User and Permission Quick Commands....432
5. Update and Troubleshooting Quick Commands....433
System update and repair....433
Logs and status....433
Process and performance quick commands....434
Boot / recovery quick commands....434
Compact One-Page Cheat List....435
Appendix B — GUI Path Reference....437
1. Settings Menu Map....437
How to open Settings....437
Core Settings map....438
Appearance....438
Network....438
Bluetooth....438
Sound....439
Displays....439
Power....439
Users....440
Sharing....440
About....440
2. App Center Quick Paths....441
How to open App Center / Ubuntu Software....441
Quick task map inside App Center....441
Search for an application....441
Browse by category....441
Install an application....441
Remove an application....441
Review software source / packaging context....442
Install Flatpak support through the GUI ecosystem....442
3. Common Admin Tasks by GUI Path....443
Update the system....443
Install software....443
Manage printer settings....443
Pair a Bluetooth device....443
Troubleshoot sound by GUI....443
Add or manage users....444

Manage startup apps.....444
Change display arrangement.....444
Change network settings.....444
Backup by GUI.....444
Inspect help.....444
4. GNOME Customization Quick Map.....445
Wallpaper and light/dark mode.....445
Themes, icons, cursor, and deeper appearance options.....445
Extensions.....445
Keyboard shortcuts.....445
Dock / panel behavior.....446
Fonts.....446
Advanced configuration.....446
5. Quick GUI Task Table.....447
Appendix C — Troubleshooting Flowcharts.....448
Flowchart 1 — No Boot.....449
Quick commands for rescue context.....450
Flowchart 2 — No Wi-Fi.....451
Quick commands.....452
Flowchart 3 — No Sound.....453
Quick commands.....454
Flowchart 4 — App Won't Install.....455
Quick commands.....456
Flowchart 5 — Black Screen After Login.....457
Quick commands.....458
Flowchart 6 — Printer Not Detected.....459
Quick commands.....460
Optional print test.....460
One-Page Compact Flow Summary.....461
Appendix D — Desktop Migration Guide.....462
1. Windows to Ubuntu Desktop Workflow Mapping.....462
Windows to Ubuntu quick map.....463
What feels different for Windows users?.....464
Practical Windows-to-Ubuntu migration advice.....464
2. macOS to Ubuntu Desktop Workflow Mapping.....465
macOS to Ubuntu quick map.....465
What feels familiar to macOS users?.....466
What feels different for macOS users?.....466
Practical macOS-to-Ubuntu migration advice.....466
Screenshot notes:.....467
3. Common Terminology Translation.....468
General desktop terms.....468
Filesystem and location terms.....469
Open-source ecosystem terms.....469

4. Practical Migration Strategy....470
CLI example: show the current home path for orientation....470
CLI example: launch Files and Settings from the command line....470
Screenshot notes:....470
Appendix E — Glossary....471
References....474
Author....475

Front Matter

How to Use This Book

This book is designed to help readers build confidence with **Ubuntu Desktop** step by step, from the first login screen to real-world desktop administration. It is not only a book about clicking buttons, and it is not only a book about typing commands. Instead, it connects the two worlds that matter most in Ubuntu: the **graphical desktop experience** and the **command-line foundation** behind it. That balance is essential because Ubuntu is most powerful when readers understand both how the desktop *looks* and how the system *works*. The uploaded Ubuntu Academy outline clearly places strong emphasis on practical desktop use, package management, GNOME workflows, system settings, users, networking, security, maintenance, troubleshooting, and guided labs, so this book should be used as both a learning guide and a hands-on practice manual.

A useful way to approach this book is to think of it as a training journey in three layers:

- **See it** — understand the Ubuntu Desktop interface visually
- **Try it** — repeat the workflow through guided tasks and labs
- **Understand it** — connect the visible action to the Linux logic underneath

For example, opening a folder in **Files** is a visual task. Listing that same folder with `ls` is a command-line task. Knowing that both refer to the same underlying filesystem is where real understanding begins. This book encourages that kind of connection throughout.

Readers do **not** need to memorize everything in one pass. In fact, Ubuntu is learned best through repetition. A student may first read a chapter to get the big picture, then repeat the steps in a virtual machine, then return to the same chapter later with new understanding. That is normal. Linux learning is often layered: what looks simple at first becomes deeper later, and what looks difficult at first becomes natural with use.

Recommended way to read and practice

Use the book in the following rhythm:

- **First pass:** read for understanding
- **Second pass:** repeat the examples on a practice system
- **Third pass:** adapt the examples to your own needs

This pattern is powerful because Ubuntu skills are not built by reading alone. They are built by doing, checking results, making small mistakes safely, and learning how to recover.

How different readers can use this book

This book is intentionally useful for several types of readers:

- **Students** who are new to Linux and need a structured introduction
- **Practitioners** who want a practical Ubuntu Desktop guide for real work
- **Teachers and lab instructors** who need repeatable examples and exercises
- **Office and home users** who want a stable and productive Linux desktop
- **Aspiring administrators** who want a desktop-focused bridge into deeper Linux skills

A beginner can follow the GUI examples first and only gradually move into Terminal use. A more advanced reader can move quickly through the visual orientation sections and focus on package management, troubleshooting, permissions, networking, and recovery.

Study strategy for best results

A strong learning habit for this book is:

- read one section,
- test one small activity,
- verify the result,
- write down what changed,
- then continue.

This habit turns passive reading into operational skill.

Illustration

Imagine Ubuntu Desktop as a modern workshop. The **GUI** is the front workbench where tools are easy to reach. The **Terminal** is the tool cabinet in the back: more powerful, more precise, and often faster once you know how to use it. A good Ubuntu user learns both spaces.

Suggested reading flow

Although the book can be used as a reference, most readers will benefit from this order:

- **Front Matter** for learning strategy and lab readiness
- **Part I** for Ubuntu Desktop foundations and desktop concepts
- **Part II** for installation and first-time setup
- **Part III–VIII** for applications, customization, administration, devices, networking, security, and maintenance
- **Part IX** for troubleshooting and recovery
- **Part X–XI** for community engagement, best practices, and guided labs

Screenshot notes

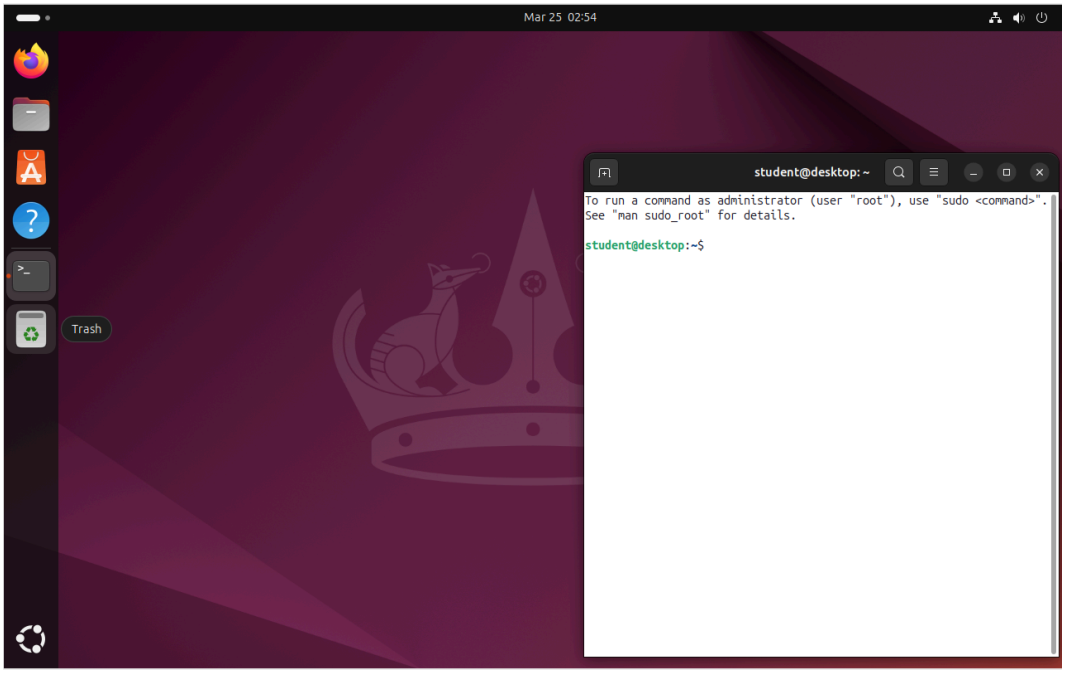


Figure. Ubuntu Desktop

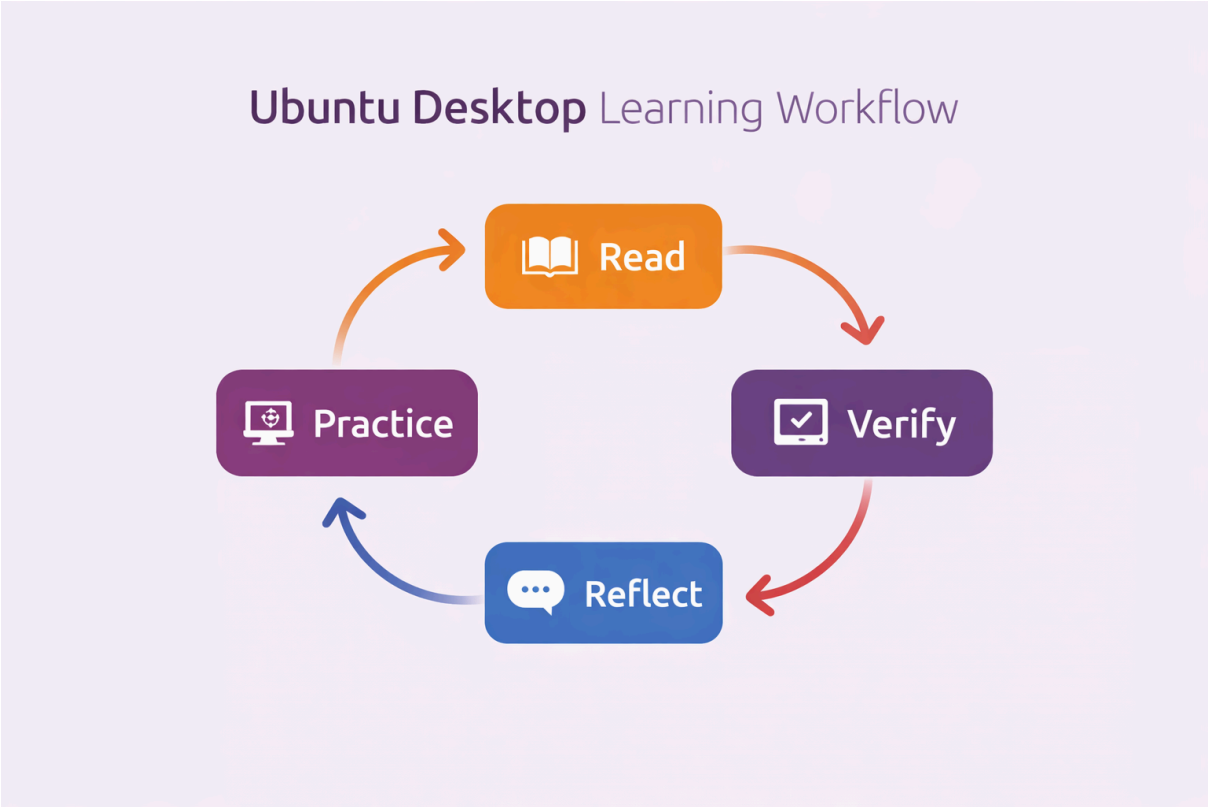


Figure. Ubuntu Desktop Learning Workflow

Lab Environment and Practice Setup

Ubuntu Desktop becomes far easier to learn when readers practice in a safe environment. For this reason, this book should be used with a **dedicated lab setup** rather than only as theory. The Ubuntu Academy structure strongly supports practical work through installation tasks, desktop exploration, package installation, customization, networking, security, maintenance, and recovery labs. A proper practice environment allows readers to experiment without fear, which is one of the most important conditions for learning Linux well.

The safest way to begin is usually with a **virtual machine**. A VM lets the reader install Ubuntu Desktop, take snapshots, test settings, break things safely, and recover quickly. For many students, this is the ideal starting point because it creates a controlled learning space. A second option is to use **Live USB mode**, which is excellent for hardware testing and temporary exploration. A third option is a **dedicated physical machine**, especially for readers who want the full performance and feel of Ubuntu Desktop. A **dual-boot** setup can also work, but it requires more care and is usually best for readers who already understand partitioning and backup discipline.

Recommended practice environments

Option 1 — Virtual Machine

Best for:

- students
- classrooms
- first-time Ubuntu learners
- testing risky changes safely

Typical tools:

- VirtualBox
- GNOME Boxes
- VMware Workstation / Player

Suggested VM baseline:

- **2 CPU cores** minimum
- **4 GB RAM** minimum, **8 GB preferred**
- **25–40 GB virtual disk**
- 3D acceleration enabled if supported

Option 2 — Live USB Session

Best for:

- hardware compatibility testing
- checking Wi-Fi, display, audio, and touchpad support
- trying Ubuntu without changing the internal disk

Option 3 — Full Install on Physical Machine

Best for:

- serious daily-use learning
- performance testing
- battery, audio, Bluetooth, printer, and monitor practice
- real user experience

Option 4 — Dual Boot

Best for:

- readers who still need another operating system
- users transitioning gradually to Ubuntu

Caution:

- **Always back up important files first**
- **Partitioning mistakes can destroy data**

Minimum practice checklist

Before starting the labs, the reader should prepare:

- a machine or VM capable of running Ubuntu Desktop
- Ubuntu Desktop installation media or ISO
- internet access for updates and packages
- a notebook or text file for practice notes
- one folder dedicated to Ubuntu exercises
- a backup method for important work

Suggested lab folder structure

Readers should create a simple workspace early in the book so that later exercises remain organized.

CLI example:

```
mkdir -p
~/Documents/ubuntu-academy/{labs,screenshots,notes,downl
oads}
tree ~/Documents/ubuntu-academy
```

Example output:

```
/home/student/Documents/ubuntu-academy
├── downloads
├── labs
├── notes
└── screenshots
```

This small structure helps learners separate experiments, notes, downloads, and captured screenshots. It also reinforces good desktop discipline from the beginning.

Practice account recommendation

If possible, use a dedicated non-production account for learning. This reduces risk and makes it easier to experiment freely.

CLI example:

```
whoami
hostname
pwd
```

Example output:

```
student
ubuntu-lab
/home/student
```

This confirms the current user, system name, and working directory.

What to practice repeatedly

A good Ubuntu lab is not only about installation. It should also allow repeated practice of:

- opening and managing applications
- navigating Files and the Home directory
- using Terminal for basic commands
- installing and removing software
- connecting to Wi-Fi and checking network status
- changing display, sound, and power settings
- creating users and adjusting permissions
- updating the system safely
- backing up and restoring data
- recovering from simple failures

Role of screenshots in the lab

This book should encourage readers to **capture screenshots as evidence of progress**. Screenshots are useful for assignments, self-review, reports, troubleshooting logs, and classroom documentation.

Suggested screenshot activities:

- first successful Ubuntu login
- desktop overview with Dock and top bar
- Files and Terminal side by side
- Wi-Fi connected successfully
- App Center installing an application
- Settings showing display or sound configuration
- Terminal output from update and upgrade operations
- successful lab completion state

CLI example:

```
gnome-screenshot -i
```

Typical result:

```
(interactive screenshot tool opens)
```

Or, on systems using GNOME screenshot shortcuts, the user may use **Print Screen** or **Shift + Print Screen** depending on the desired capture mode.

Illustration

Think of the lab environment as a training gym. You do not go to a gym only to read about exercise. You go there to repeat movements safely until they become natural. Ubuntu learning works the same way.

Screenshot notes

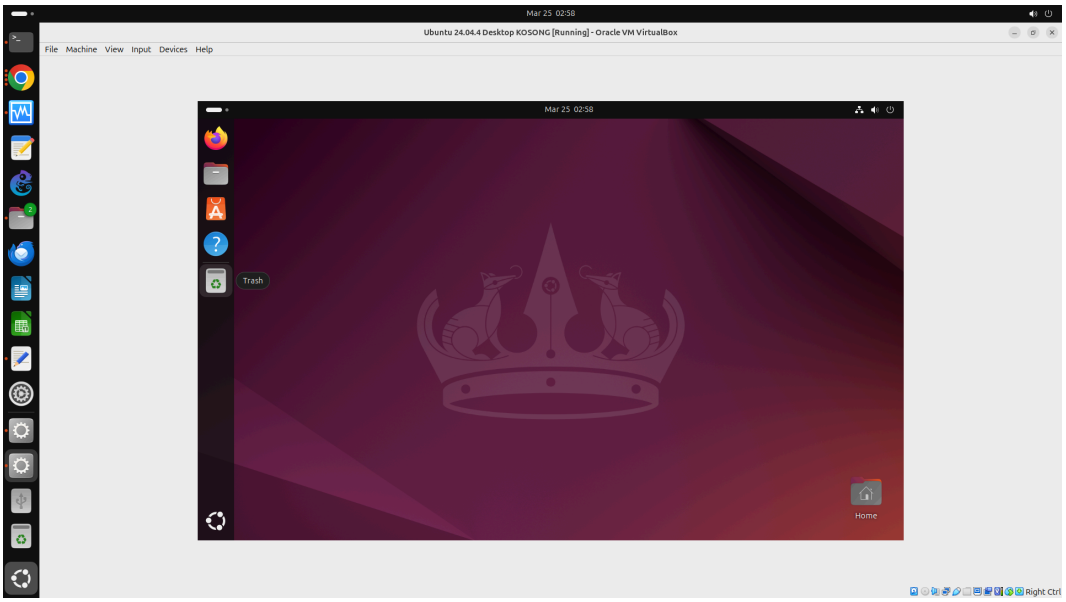


Figure. Ubuntu Desktop in VirtualBox VM

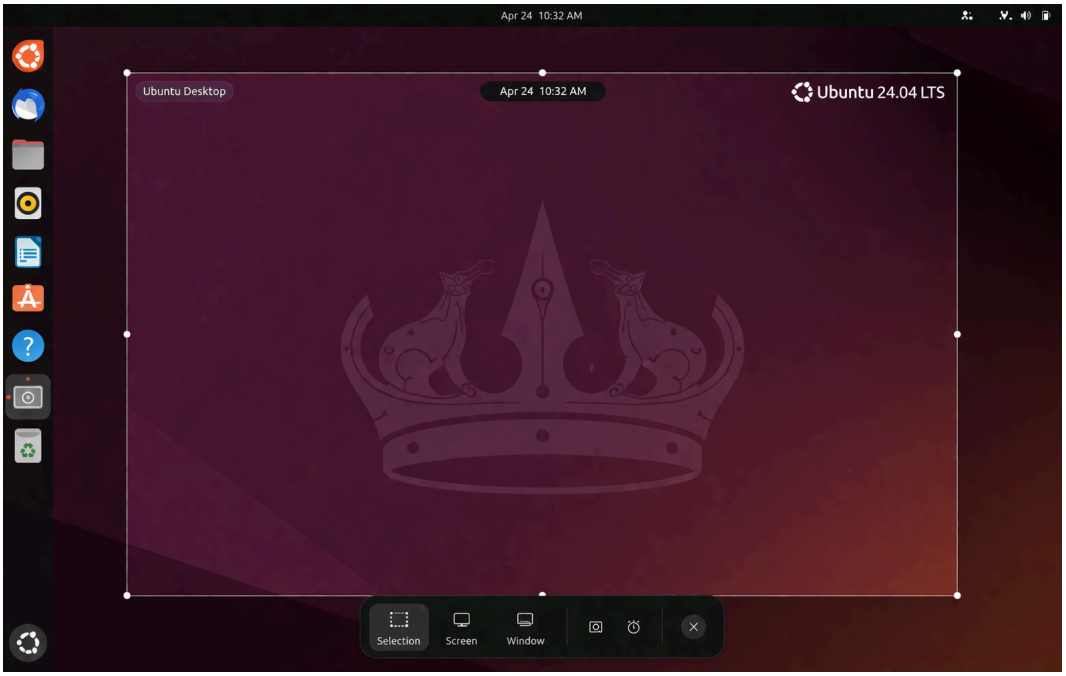


Figure. GNOME screenshot overlay active

Conventions Used in This Book

To help readers move smoothly between explanation, hands-on practice, and technical precision, this book uses a consistent set of conventions. These conventions are not merely editorial style choices. They are learning signals. They tell the reader what is conceptual, what is actionable, what must be typed exactly, what is only an example, and what requires special care. In a technical book, consistent conventions reduce confusion and increase confidence.

Typographic conventions

The following styles are used throughout the book:

- **Bold** for important concepts, warnings, and key terms the reader should remember
- *Italic* for foreign terms, emphasis, first-use terminology, and conceptual distinctions
- `monospace` for commands, file names, package names, paths, configuration values, and terminal text

Examples:

- **GNOME** is the default Ubuntu Desktop environment.
- *Wayland* and *X.Org* are display technologies discussed in the book.
- Use `sudo apt update` to refresh package metadata.
- Personal files usually live under `/home/username`.

Command-line conventions

Whenever a CLI example appears:

- text in `monospace` should be typed exactly unless stated otherwise
- lines beginning with `#` in explanations are comments, not commands to type
- example output is shown separately and may differ depending on system version, hardware, installed packages, or user name
- commands that require elevated privileges often begin with `sudo`

Example:

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Reading package lists... Done
```

Readers should understand that their own output may not match every line exactly. That is normal. What matters is understanding the **meaning** of the result.

Path conventions

Common path symbols are used throughout the book:

- `~` means the current user's home directory
- `/` refers to the root of the filesystem
- relative and absolute paths are both used where appropriate

Example:

```
cd ~/Documents
pwd
```

Example output:

```
/home/student/Documents
```

GUI path conventions

Because this is a desktop-focused book, many tasks are presented as GUI navigation paths.

Format example:

```
Settings → Wi-Fi → select network → Connect
```

or

```
Activities → search "Terminal" → open Terminal
```

This format allows readers to follow desktop workflows quickly and consistently.

Warning and tip conventions

This book should use clear learning signals such as:

- **Important** — a concept that affects later understanding
- **Note** — a useful clarification or helpful detail
- **Tip** — a shortcut, smart habit, or easier approach
- **Warning** — a risky step that may affect data, boot behavior, settings, or security

Example:

Warning: `dd` writes directly to a device. Choosing the wrong target disk can destroy data.

Tip: Use a VM snapshot before testing risky package or configuration changes.

Output variability

Readers should expect some variation in:

- package versions
- desktop session names
- hardware device names
- network interface names
- repository mirrors
- application availability by release
- terminal formatting

For example, one system may show `wayland`, while another may show `x11`:

```
echo $XDG_SESSION_TYPE
```

Example output:

```
wayland
```

Alternative output:

```
x11
```

This does not automatically indicate a problem. It indicates a different graphical session type.

Naming conventions in labs

Lab examples may use sample names such as:

- `student`
- `ubuntu-lab`
- `example.txt`
- `~/Documents/ubuntu-academy`
- `wlp2s0` for Wi-Fi interface examples

Readers should replace these with their own real values where needed.

Screenshot code conventions

The book includes explicit screenshot production notes so that the manuscript can later be illustrated consistently. These use the following format:

```
[Screenshot-CODE: description of what should be visible]
```

This is especially useful for desktop manuals because screenshots reinforce spatial memory and reduce beginner confusion.

Example:

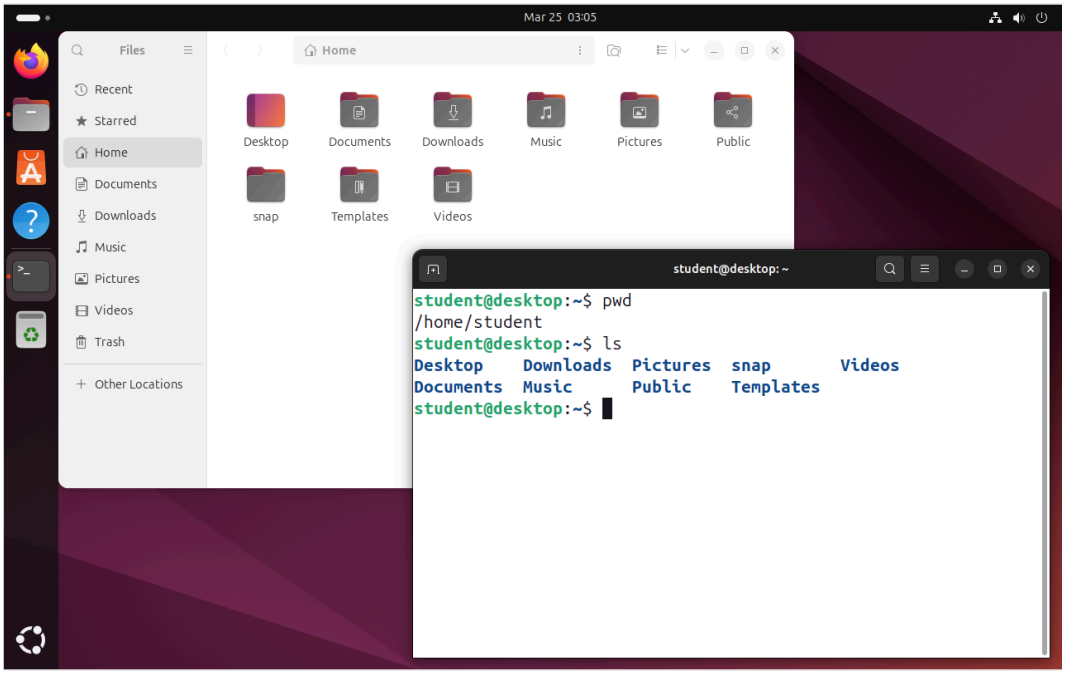


Figure. Desktop vs.CLI / Terminal

Illustration

> Think of these conventions as road signs in a campus. Without road signs, the campus still exists, but movement is slower and more confusing. With road signs, readers can focus on learning instead of guessing what each format means.

Learning Outcomes

> By the end of this book, readers should not only recognize Ubuntu Desktop visually, but also use it confidently for study, work, customization, maintenance, and troubleshooting. The Ubuntu Academy outline clearly suggests a performance-oriented learning path that includes desktop navigation, installation, applications, software management, GNOME customization, users, permissions, devices, networking, security, maintenance, recovery, open-source participation, and guided labs. The learning outcomes below therefore reflect both conceptual understanding and practical operational skill.

General learning outcome

After completing this book, the reader will be able to **install, use, customize, secure, maintain, and troubleshoot Ubuntu Desktop in realistic personal, educational, and professional contexts**.

Specific learning outcomes

By the end of the book, readers should be able to:

1. Understand Ubuntu Desktop as a platform

- explain what Ubuntu Desktop is and how it differs from Ubuntu Server
- describe the role of Linux, GNU tools, Debian heritage, and open-source principles
- distinguish between LTS and interim releases
- recognize the role of GNOME in Ubuntu Desktop

2. Navigate the Ubuntu Desktop environment

- identify the top bar, Dock, Activities Overview, App Grid, Files, Settings, and Quick Settings
- use search, notifications, calendar, and workspace navigation effectively
- apply keyboard shortcuts for faster desktop productivity
- understand accessibility options for inclusive desktop use

3. Prepare and install Ubuntu Desktop

- evaluate hardware suitability for Ubuntu Desktop
- create bootable installation media using GUI or CLI approaches
- use Live USB mode for testing hardware compatibility
- perform a clean install, VM install, or guided dual-boot workflow
- complete first-boot configuration and initial updates safely

4. Manage applications and software sources

- install, update, remove, and search for software through the App Center and Terminal
- understand package sources such as APT, Snap, Flatpak, and `.deb` packages
- compare GUI and CLI workflows for software management
- verify application installation and basic package state

5. Use Ubuntu Desktop productively in daily work

- manage files and folders confidently using both Files and Terminal
- organize work through workspaces, tiling, search, and shortcut usage
- use core desktop applications for writing, browsing, media, and communication
- understand the practical workflow differences between Ubuntu and other desktop systems

6. Customize and optimize the desktop

- adjust themes, appearance, Dock behavior, extensions, fonts, and desktop preferences
- configure GNOME in ways that improve focus and productivity
- make sensible customization decisions without compromising stability

7. Manage users and permissions

- create and manage user accounts through GUI and CLI methods
- understand file ownership and permission basics
- apply safe privilege use with `sudo`
- distinguish personal user tasks from administrative tasks

8. Configure hardware, devices, and multimedia

- verify hardware detection for displays, audio, printers, scanners, Bluetooth devices, and removable media
- troubleshoot common desktop device issues
- manage external displays and practical workstation hardware workflows

9. Configure networking for desktop users

- connect to Wi-Fi and wired networks
- inspect IP addressing and connectivity from GUI and CLI tools
- configure basic networking options relevant to desktop use
- understand the role of DNS, proxy settings, and simple firewall use

10. Secure and maintain Ubuntu Desktop

- apply good password, update, backup, and privacy habits
- understand the purpose of the firewall, updates, and safe software sources
- perform routine maintenance tasks
- prepare and verify backup procedures

11. Troubleshoot and recover from problems

- identify and respond to common Ubuntu Desktop issues
- use logs, system tools, and simple CLI checks during troubleshooting
- understand basic boot and recovery workflows
- use Live USB and recovery-oriented thinking when the system fails to start normally

12. Participate in the Ubuntu ecosystem

- identify support and learning channels such as documentation and community forums
- ask better technical questions
- understand that Ubuntu is part of a broader open-source culture
- develop disciplined habits as a responsible Ubuntu Desktop user

Performance-oriented outcome

A strong reader should finish the book able to do more than describe Ubuntu. They should be able to:

- build a usable Ubuntu Desktop environment,
- keep it updated and productive,
- solve routine problems,
- document what they changed,
- and continue learning independently.

Illustration

A useful way to frame the learning outcomes is this: by the end of the book, the reader should move from being a **visitor** in Ubuntu Desktop to being a **capable resident**. At first, they only recognized the streets. Later, they know how to work, fix, organize, and navigate confidently.

Ubuntu Desktop Competency Map

A competency map helps readers see that Ubuntu Desktop mastery is not one single skill. It is a collection of related competencies that build on one another. The Ubuntu Academy outline already organizes the book into a strong progression: foundations, installation, applications, customization, administration, devices, networking, security, troubleshooting, ecosystem participation, and guided labs. That structure can be translated into a practical competency map so readers, instructors, and institutions can see how beginner tasks grow into professional desktop capability.

Competency philosophy

This book treats Ubuntu Desktop competence as a ladder:

- **Awareness** — recognize the environment and basic concepts
- **Operation** — perform routine tasks correctly
- **Control** — configure, customize, and maintain the system
- **Diagnosis** — investigate and solve problems
- **Independence** — continue learning and using Ubuntu effectively in real contexts

Domain-based competency map

Domain 1 — Ubuntu Desktop Foundations

Readers should be able to:

- explain what Ubuntu Desktop is
- describe GNOME and basic desktop architecture
- distinguish desktop workflows from server workflows
- understand release choices and desktop flavor context

Typical evidence:

- can explain LTS vs interim releases
- can identify GNOME components on screen
- can describe Wayland vs X.Org at a beginner level

Domain 2 — Desktop Navigation and Daily Use

Readers should be able to:

- log in and move confidently through the interface
- use top bar, Dock, Activities, search, App Grid, and Files
- organize windows and workspaces
- use keyboard shortcuts productively

Typical evidence:

- opens apps by search and shortcut
- manages files in Home, Documents, Downloads, and Pictures
- uses multiple workspaces to separate tasks

Domain 3 — Installation and First-Time Setup

Readers should be able to:

- prepare installation media
- test Ubuntu in Live Mode
- install Ubuntu Desktop in a VM or physical system
- complete first-boot checks and updates

Typical evidence:

- creates a bootable USB or VM
- confirms hardware support in a live session
- updates the system after installation

Domain 4 — Software Management

Readers should be able to:

- install and remove applications
- understand APT, Snap, Flatpak, and `.deb`
- compare GUI software installation with CLI software installation
- verify package state and update status

Typical evidence:

- installs an application from App Center
- installs a package with `apt`
- recognizes the source of a package

Domain 5 — Customization and User Experience

Readers should be able to:

- personalize GNOME sensibly
- adjust themes, Dock settings, wallpaper, and extensions
- optimize the workspace for productivity

Typical evidence:

- changes appearance intentionally
- improves usability without making the system unstable
- understands the trade-off between beauty and performance

Domain 6 — Users, Permissions, and Local Administration

Readers should be able to:

- create and manage users
- understand file permissions and ownership basics
- use `sudo` correctly
- perform everyday administrative tasks safely

Typical evidence:

- creates or modifies a user account
- identifies file ownership with `ls -l`
- understands when administrative rights are required

Domain 7 — Devices, Multimedia, and Peripherals

Readers should be able to:

- connect printers, removable drives, and Bluetooth devices
- configure sound and displays
- inspect hardware detection

Typical evidence:

- adds a printer
- changes audio output device
- confirms hardware using GUI or CLI tools

Domain 8 — Networking for Desktop Users

Readers should be able to:

- connect to Wi-Fi and wired networks
- verify connectivity
- understand desktop-relevant network settings

Typical evidence:

- connects successfully to Wi-Fi
- checks IP information
- tests connectivity with `ping`

Domain 9 — Security, Privacy, and Maintenance

Readers should be able to:

- keep Ubuntu updated
- use safe software habits
- understand backup and restore basics
- apply simple firewall and privacy practices

Typical evidence:

- updates regularly
- uses trusted package sources
- configures backup tools
- understands why maintenance is part of security

Domain 10 — Troubleshooting and Recovery

Readers should be able to:

- diagnose common desktop issues
- perform basic log and process checks
- use recovery-oriented thinking
- work with Live USB and boot repair concepts

Typical evidence:

- identifies whether a problem is app-level, desktop-level, or boot-level
- performs safe checks before risky repair
- preserves user data first

Domain 11 — Community and Continuing Growth

Readers should be able to:

- locate help
- ask useful questions
- read documentation
- contribute responsibly to the Ubuntu ecosystem

Typical evidence:

- uses official docs and community help channels
- writes clearer problem descriptions
- continues learning beyond the book

Competency progression table

Level	Reader Profile	What They Can Do
1 — Explorer	New user	Log in, open apps, browse files, use search, understand basic layout
2 — Operator	Daily desktop user	Install apps, manage files, update system, connect devices and networks
3 — Configurator	Confident practitioner	Customize GNOME, manage users, adjust settings, maintain a stable workstation
4 — Troubleshooter	Capable support user	Diagnose common problems, use CLI checks, recover from routine failures
5 — Independent Ubuntu User	Advanced learner / junior admin	Deploy, maintain, secure, document, and improve Ubuntu Desktop workflows independently

How instructors can use this map

The competency map can also support:

- classroom assessment
- lab sequencing
- practical exams
- self-check milestones
- institutional desktop training plans

Illustration

Imagine the competency map as a metro map for Ubuntu Desktop. Each line represents a skill domain. Some stations are easy and familiar, like opening Files or changing Wi-Fi. Other stations are more advanced, like package sources, permissions, firewall rules, or boot recovery. The purpose of the book is not to throw readers into the whole city at once. It is to help them travel the network with confidence.

Part I — Foundations of Ubuntu Desktop

Chapter 1 — Understanding Ubuntu Desktop

Ubuntu Desktop is more than just an operating system that happens to run on a PC. It is a **complete desktop computing environment** built on the Linux kernel, packaged as a user-friendly distribution, and supported by a large open-source ecosystem. In the Linux world, Ubuntu is often described as a *distribution* or *distro*, because it does not only provide the kernel itself, but also the graphical desktop, system tools, applications, package repositories, and a consistent update model for everyday use. This is one of the reasons Ubuntu became such an important entry point for students, professionals, and new Linux users: it combines the technical power of Linux with a much more approachable desktop experience.

To understand Ubuntu clearly, it helps to separate a few terms that are often mixed together. **Linux** is technically the kernel, the core software that communicates with hardware and manages system resources. On top of that kernel, a usable system also needs command-line tools, libraries, services, installers, package managers, and often a graphical interface. This broader collection is commonly described as *GNU/Linux*, because many of the standard user-space tools come from the GNU project. Ubuntu stands on this foundation and transforms it into a polished desktop platform that can be installed, updated, customized, and used productively by ordinary users. In simple terms, Linux is the engine, while Ubuntu is the fully assembled vehicle.

Ubuntu also belongs to an important family tree in the Linux ecosystem. It is **based on Debian**, a distribution widely respected for stability, software packaging, and disciplined release practices. Debian uses the `.deb` package format and established a strong reputation for reliable software management. Ubuntu builds on that base and adds easier installation, more convenient graphical tools, stronger desktop focus, and a smoother experience for people who want Linux without having to assemble everything by hand. This relationship is one of Ubuntu's biggest strengths: Ubuntu benefits from Debian's technical foundation while making Linux more accessible to a wider audience.

Another essential idea behind Ubuntu is **open source**. Ubuntu exists because the Linux kernel, GNU tools, and thousands of other programs are developed under licenses that allow people to use, study, modify, and share software. For the learner, this means freedom. For the institution, it means transparency and flexibility. For the practitioner, it means access to a vast ecosystem of tools without being tied to one vendor. Open source is not only a licensing model; it is also a culture of collaboration, documentation, peer review, and continuous improvement. Ubuntu makes that culture visible in a practical, everyday form. When a student installs Ubuntu on a laptop, they are not only installing software; they are stepping into a global learning community.

A useful way to picture Ubuntu is to imagine a university campus. The **Linux kernel** is the land and infrastructure. **GNU tools and system libraries** are the utilities,

roads, and buildings. **Debian** is the original city plan that organizes everything carefully. **Ubuntu** is the well-designed campus version built for easier navigation, comfort, and daily use. The result is a system that remains powerful inside, but feels much more welcoming on the surface.

Ubuntu Desktop vs Ubuntu Server

Although Ubuntu Desktop and Ubuntu Server come from the same family and use many of the same repositories, they are designed for very different purposes. **Ubuntu Desktop** is intended for interactive human use. It includes a graphical user interface, desktop applications, file browsing tools, multimedia capabilities, and settings panels that help users work comfortably in a visual environment. A typical Ubuntu Desktop installation supports browsing, writing, media playback, email, communication, and productivity tasks out of the box.

Ubuntu Server, in contrast, is designed primarily to provide services rather than a full graphical workspace. It usually focuses on command-line administration and is commonly used for web hosting, databases, file services, remote access, and infrastructure roles. In many cases, a server does not need a graphical environment at all. This lighter design saves resources and keeps the system focused on stability, automation, and remote management. While both Desktop and Server share the same Ubuntu roots, the daily experience of using them is very different. Desktop is for direct human interaction; Server is for delivering services efficiently and reliably.

A simple illustration may help:

- **Ubuntu Desktop** is like a personal study desk with monitor, browser, office tools, music player, settings panels, and visual notifications.
- **Ubuntu Server** is like a machine room in the background: no decoration, no casual interaction, but extremely important for delivering services to others.

For this book, the focus is clearly on **Ubuntu Desktop**: the environment where users learn, create, communicate, browse, write, manage files, and explore Linux visually and interactively. That focus is also reflected in the Ubuntu Academy outline, which emphasizes GNOME, desktop features, package management, user experience, customization, and practical administration skills.

Ubuntu Release Model: LTS vs Interim Releases

Ubuntu follows a release model that helps users choose between **stability** and **newness**. Canonical publishes Ubuntu on a predictable schedule, typically twice a year. These regular releases are often called *interim releases*. They introduce newer software, updated kernels, and newer hardware support more quickly. For users who want fresh features and do not mind upgrading more frequently, interim releases can be exciting and useful.

Alongside these, Ubuntu also provides **LTS — Long Term Support** releases. LTS versions are designed for users and organizations that prioritize long-term reliability,

consistency, and lower maintenance pressure. In practice, LTS releases are usually the best choice for classrooms, offices, labs, and most daily-use workstations, because they reduce the need for frequent major upgrades while still receiving maintenance and security updates. They are especially valuable in environments where stability matters more than having the very newest feature immediately.

A practical way to think about it is this:

- **LTS** is like choosing a dependable textbook edition used for several academic years.
- **Interim releases** are like following the newest edition every semester to get the latest changes, even if that means adjusting more often.

For beginners, students, and institutional deployments, **LTS is usually the recommended path**. It offers a calmer learning environment, fewer disruptions, and better predictability. For enthusiasts, testers, or users with newer hardware needs, interim releases may be attractive because they bring newer components sooner. Either way, Ubuntu's structured release cycle is one of its strengths: it gives users a clear roadmap instead of forcing everyone into the same pace.

Official Ubuntu Flavors and Desktop Choices

One of the most inspiring aspects of Ubuntu is that there is not only one visual style or desktop approach. Ubuntu offers **official flavors**, which are alternative editions built on the same Ubuntu foundation but with different desktop environments, tool selections, and user experiences. These flavors are officially recognized, released on the same overall cycle, and supported as part of the broader Ubuntu ecosystem.

This matters because desktop computing is personal. Some users want a modern, elegant, and streamlined interface. Others want something lightweight for older hardware. Some prefer a traditional desktop metaphor, while others want a more customizable workspace. Ubuntu's flavor ecosystem responds to these different needs without forcing users to abandon the Ubuntu base.

A few examples illustrate this choice:

- **Ubuntu Desktop** uses **GNOME** by default and focuses on a clean, modern workflow.
- **Kubuntu** uses KDE Plasma and offers a richer, more configurable desktop style.
- **Ubuntu Studio** targets creative users who work with audio, video, and graphics.
- **Edubuntu** is designed with educational settings in mind.

For this book, the main focus remains the **default Ubuntu Desktop experience with GNOME**, because that is the standard environment most learners will encounter first, and it aligns with the Ubuntu Academy structure. GNOME has become the default desktop for Ubuntu and is designed to feel modern, elegant, and productive,

especially for users moving between laptop, desktop, and other contemporary computing environments.

An easy illustration is to imagine Ubuntu as a campus with multiple buildings serving different learning styles:

- one building is optimized for **general study and productivity**,
- another for **creative studios**,
- another for **older, lighter machines**,
- another for **education labs**.

The campus is still Ubuntu, but each building is arranged for a different kind of user.

Why Ubuntu Desktop for Learning, Work, and Daily Use

Ubuntu Desktop is particularly strong because it works well in three important roles at once: **learning platform, productivity platform, and daily personal workstation**. For students, Ubuntu is a practical way to learn how modern operating systems, software repositories, permissions, updates, and desktop environments work. It encourages hands-on learning without hiding the system completely. Users can start with graphical tools and gradually become comfortable with the terminal when needed.

For professionals, Ubuntu provides a stable and capable work environment. It supports office suites, browsers, multimedia, communication tools, and development workflows, while also offering access to a massive software ecosystem through repositories and package tools. The fact that it is both beginner-friendly and professionally respected makes it especially valuable in education and training. A learner can begin with simple tasks like browsing files and installing applications, then later grow into scripting, troubleshooting, and system administration without leaving the platform.

For daily use, Ubuntu Desktop is attractive because it is:

- **free to use**
- **customizable**
- **secure by design**
- backed by a large open-source community
- suitable for web, office, media, communication, and study tasks

That combination makes Ubuntu especially meaningful in schools, laboratories, community technology centers, small offices, and personal laptops. It allows users to build useful computing environments without being locked into a single vendor model. At the same time, it provides enough depth to remain interesting and challenging as the user grows.

A good mental picture is this: Ubuntu Desktop is like a bicycle that is easy enough for a beginner to ride, but strong and adaptable enough for long-distance travel. You do not outgrow it quickly. Instead, you keep discovering what else it can do.

Community, Documentation, and Support Channels

A major strength of Ubuntu is that learning does not stop at installation. Ubuntu is supported by a wide network of **documentation, discussion spaces, community help channels, and contribution pathways**. The Ubuntu Academy outline itself highlights the importance of understanding governance, finding help, interpreting licenses, and participating in the ecosystem through platforms such as documentation sites, community forums, and question-and-answer spaces.

This support ecosystem matters because no one learns Linux alone. At some point, every user asks questions such as:

- Why is my Wi-Fi not working?
- Which release should I install?
- How do I update safely?
- Why does this package fail?
- Which desktop flavor fits my laptop best?

Ubuntu is powerful partly because these questions usually have an answer somewhere in the ecosystem. Official documentation gives structured guidance. Community forums and discussion platforms provide shared practical experience. Q&A communities help solve focused technical issues. Bug-reporting systems allow users to improve the platform rather than only consume it. In this sense, Ubuntu is not just software; it is also a collaborative learning network.

For students and practitioners, this creates a healthy mindset: **using Ubuntu means learning how to learn**. Instead of waiting passively for a vendor to solve every problem, users become more capable of reading documentation, comparing solutions, testing ideas, and contributing back. That habit is one of the most valuable lessons Ubuntu can teach.

Chapter Reflection

By the end of this chapter, readers should understand that Ubuntu Desktop is not merely “Linux with a pretty screen.” It is a carefully assembled desktop distribution built on the Linux kernel, strengthened by Debian’s packaging tradition, guided by open-source principles, and shaped by a strong user and developer community. It differs from Ubuntu Server in purpose and experience, offers clear choices through LTS and interim releases, supports different user needs through official flavors, and remains highly relevant for learning, work, and daily digital life.

Key Takeaways

- **Ubuntu is a Linux distribution**, not just a kernel.
- **Ubuntu builds on Debian**, adding accessibility and a stronger desktop experience.
- **Open source** gives users freedom, transparency, and collaboration opportunities.
- **Ubuntu Desktop** is designed for interactive human use, while **Ubuntu Server** is optimized for service delivery.
- **LTS releases** favor long-term stability; **interim releases** favor newer features.
- **Official Ubuntu flavors** provide different desktop experiences for different needs.
- **Ubuntu Desktop is excellent for learning, work, and everyday productivity**.
- **Community and documentation are part of the Ubuntu experience**, not an afterthought.

Chapter 2 — Linux Desktop Technology in Context

To use Ubuntu Desktop with confidence, readers need more than a list of clicks and menus. They need a simple mental model of **how a Linux desktop actually works**. This chapter introduces the technology behind the visual experience of Ubuntu Desktop and explains why Ubuntu feels smooth, flexible, and adaptable on many kinds of hardware. In the Ubuntu Academy context, this chapter supports later topics such as customization, troubleshooting, performance tuning, and choosing the right working environment for real users.

A helpful way to begin is with this idea: a Linux desktop is not one single monolithic object. It is a **collection of components** that work together to create the graphical environment that users see every day. Instead of treating the desktop as a mysterious black box, Ubuntu allows us to understand it as layers: the display system, the window manager, the desktop environment, the file manager, the settings tools, and the application toolkit. Once these layers become clear, Ubuntu Desktop stops feeling “complicated” and starts feeling beautifully logical.

Imagine a modern office building. The **display server** is the electricity and wiring that make the building usable. The **window manager** is the staff member arranging rooms and doors so everything stays organized. The **desktop environment** is the interior design and workflow layout. The **file manager** is the records room that keeps documents accessible. The **settings framework** is the control panel for lights, temperature, and access rules. The **toolkit** is the standard furniture and equipment used across all rooms so they feel consistent. This illustration is useful because a Linux desktop really is a coordinated system rather than a single program.

What a Linux Desktop Environment Is

A **desktop environment** is the graphical layer that lets users interact with Linux using windows, icons, menus, panels, folders, wallpapers, and visual applications instead of relying only on text commands. Without a desktop environment, Linux can still run perfectly well, but the user would mostly work through a command-line interface. With a desktop environment, the system becomes far more approachable for students, office users, and general desktop workflows.

In practical terms, a desktop environment combines several essential parts into one coherent user experience. It usually includes the visual shell, a way to manage application windows, a set of core utilities, consistent themes, and applications designed to work well together. This is why a desktop environment influences not only how Ubuntu looks, but also how it feels to use every day. The placement of panels, the style of system menus, the file manager, the default text editor, and even notification behavior are all shaped by the desktop environment.

This modular design is one of Linux’s most interesting strengths. Unlike some operating systems where the graphical shell is tightly fused into the platform, Linux allows different desktop environments to be used on the same system. That flexibility means users can prioritize modern design, familiar workflow, lower memory usage, or

better responsiveness on older hardware. Ubuntu benefits greatly from this model because it can offer a polished default while still allowing alternatives when needed.

In simple terms, a desktop environment answers three questions:

- **How does the system look?**
- **How does the user interact with it?**
- **Which core desktop tools are included by default?**

GNOME as the Default Ubuntu Desktop

Ubuntu Desktop uses **GNOME** as its default desktop environment. GNOME is known for presenting a modern, clean, and productivity-focused interface. Rather than overwhelming users with too many visible controls, GNOME emphasizes a simpler working model built around the Activities overview, search, application launching, workspace navigation, and focused window management. This design makes Ubuntu Desktop approachable for newcomers while still being efficient for experienced users.

GNOME in Ubuntu is not just a pretty interface. It is a practical desktop framework that supports everyday learning and work. Core actions such as opening files, switching applications, changing settings, browsing folders, managing notifications, or moving between workspaces are designed to feel consistent and discoverable. For many users, this is one of Ubuntu's biggest advantages: the system remains technically powerful, but the daily workflow does not require memorizing dozens of commands from the start.

Ubuntu also adds its own refinements to the GNOME experience. Readers may notice that the Ubuntu session feels slightly different from a plain upstream GNOME session because Ubuntu includes its own defaults, visual adjustments, and workflow choices. Even so, the overall GNOME philosophy remains visible: **search-driven navigation, organized workspaces, strong keyboard usage, and a distraction-reduced layout**.

A simple illustration may help. Traditional desktops often behave like a desk with many drawers left open at once. GNOME behaves more like a clean worktable where tasks are grouped, search is fast, and switching context is deliberate. That is why GNOME often feels calm and modern, especially on laptops and productivity-focused machines. This aligns well with Ubuntu Academy's emphasis on workflow optimization, keyboard shortcuts, workspaces, and practical desktop administration.

Key GNOME ideas readers should remember:

- **Activities overview** as the central launch and navigation space
- **Workspaces** for organizing tasks without clutter
- **Files (Nautilus)** as the standard file manager
- **Settings** as the main control hub
- **Extensions and tweaks** for advanced customization later on

Wayland vs X.Org in Ubuntu Desktop

One of the most important background technologies in a Linux desktop is the **display system**. In older Linux desktops, this role was usually handled by the **X Window System**, often called *X* or *X.Org*. More recently, **Wayland** became the default display protocol in Ubuntu. This change matters because the display system affects how windows are drawn, how input devices are handled, how screen sharing works, and how smoothly desktop graphics behave.

The X Window System has a long and important history. It predates Linux itself and was built as a lightweight graphical framework with network-oriented design ideas. In the X model, the X server runs locally and manages the display, keyboard, and mouse, while applications connect to that server as clients. This design was powerful and influential, but it also came from a very different era of computing. Modern desktop expectations such as smoother compositing, tighter application isolation, and more secure screen handling encouraged the move toward newer approaches.

Wayland is that newer approach. In Ubuntu, Wayland became the default because it offers a cleaner, more modern foundation for graphical desktops. It is closely associated with better security boundaries and better integration with newer desktop workflows. Related improvements such as PipeWire also help restore or improve things like screen sharing and desktop recording in modern Linux environments. For most users, Wayland works very well and simply feels normal because Ubuntu handles the complexity behind the scenes.

However, it is important to be realistic. Not every graphics driver, hardware setup, or older application behaves perfectly in every situation. Because of that, Ubuntu still gives users the option to choose **GNOME on Xorg** if they encounter compatibility issues. This is especially useful when dealing with older hardware, specific graphics drivers, or software that still behaves better under X. In real-world desktop administration, this flexibility is valuable: the goal is not to worship one technology, but to choose the one that works best for the user's environment.

A practical comparison is useful:

- **X.Org** is like a very experienced older transport system: flexible, proven, and widely supported, but built for an earlier era.
- **Wayland** is like a newer transport network: cleaner design, better efficiency for modern needs, and stronger safety controls, though a few older routes may still be more comfortable on the older system.

Core Desktop Components

To understand Ubuntu Desktop clearly, readers should become familiar with the core parts that make the visual environment possible. These components are not just technical trivia. They explain why customization works the way it does, why some applications look more integrated than others, and why certain performance or compatibility issues appear.

Display server

The **display server** is the layer that connects graphical applications to the user's screen, mouse, and keyboard. It provides the foundation on which windows appear and respond to input. In older systems, this role is handled by X.Org; in current Ubuntu Desktop, it is usually handled by Wayland. Without a display server, applications would have no standard visual path to draw windows or receive graphical input.

Window manager

The **window manager** controls how application windows are placed, moved, resized, decorated, minimized, maximized, and stacked. If the display server is the stage, the window manager is the person organizing where every actor stands. It determines how the desktop feels when multiple applications are open at once. Borders, title bars, placement behavior, snapping, and general window organization all depend on this layer.

File manager

The **file manager** is the visual tool used to browse folders, copy files, move documents, access removable drives, and connect to network shares. On Ubuntu's GNOME desktop, this is typically **Files**, also known historically as **Nautilus**. For many users, the file manager is one of the most frequently used desktop tools because it bridges the abstract filesystem and the user's daily tasks. It turns folders and paths into something readable and navigable.

Settings framework

A modern desktop also needs a coherent **settings framework**. This is the part that lets users configure network connections, displays, sound, power behavior, online accounts, appearance, input devices, and many other system preferences in an organized way. In Ubuntu Desktop, the Settings application acts as a central control panel, while advanced tweaks may also involve specialized tools such as GNOME Tweaks or lower-level configuration systems introduced later in the book. This layered design makes Ubuntu accessible for beginners while still giving advanced users room to fine-tune behavior.

Widget and toolkit basics

A desktop's visual elements—buttons, scroll bars, icons, menus, text fields, and many other user interface parts—are usually created through a **toolkit**. A toolkit is a collection of libraries and reusable components that help applications look and behave consistently. In Linux desktops, two major toolkit families are often discussed: **GTK** and **Qt**. Applications built with the toolkit that matches the desktop environment usually feel more native and visually integrated, although cross-toolkit applications can still run.

This may sound abstract, but it has visible results. A GTK-based application on GNOME often looks more naturally at home, while a Qt-based application may bring a slightly different visual style unless themes are adjusted. That is why desktop consistency is not only about wallpaper or icons; it is also about the underlying toolkit choices that shape the appearance of application controls.

Readers should remember these core building blocks:

- **Display server** → draws and handles graphical display/input
- **Window manager** → arranges and controls windows
- **File manager** → provides visual access to files and folders
- **Settings framework** → centralizes system configuration
- **Toolkit** → gives applications consistent interface elements

Workspaces, Notifications, and Modern Desktop Workflows

Modern desktop productivity is not only about opening applications. It is about organizing attention. Ubuntu's GNOME-based workflow supports this through **workspaces**, **notifications**, **search**, and efficient keyboard navigation. These features are part of what makes Ubuntu Desktop feel modern rather than cluttered. Instead of forcing every task into one crowded screen, GNOME encourages users to separate activities into cleaner visual contexts.

Workspaces are especially important. A workspace is like having multiple virtual desks on the same computer. A user might keep a web browser on one workspace, a document editor on another, and a file manager or communication tool on a third. This reduces visual overload and improves focus. Ubuntu's default keyboard shortcuts make workspace switching fast, and this supports a workflow that is both simple and powerful for students, writers, developers, administrators, and office users.

Notifications also matter. A good desktop should inform the user without constantly interrupting them. Ubuntu integrates notifications into the desktop in a way that supports awareness while preserving concentration. Features such as *Do Not Disturb* and online account integration show that the desktop is no longer only about local files and windows; it is also about calendars, messages, cloud services, and modern productivity habits.

A simple study illustration is useful here. Think of workspaces as separate pages in a notebook. One page is for reading, one for writing, one for file organization. Notifications are like sticky notes attached to the edge: useful when needed, distracting when excessive. GNOME's modern workflow tries to keep these tools available but controlled. That design fits Ubuntu Academy's emphasis on workflow optimization, screenshots, file search, keyboard shortcuts, and practical daily use.

Useful GNOME workflow habits include:

- using the **Super** key to open Activities
- switching between virtual desktops with workspace shortcuts
- tiling windows for side-by-side work
- using **Alt+Tab** to move quickly between active applications
- relying on search instead of hunting through long menus

Choosing the Right Desktop for Performance, Usability, and Hardware Constraints

Although Ubuntu Desktop uses GNOME by default, not every machine or user has the same needs. Some users value polish and integrated workflow. Others need a lighter environment because their system has limited memory, older graphics hardware, or battery constraints. This is where Linux's desktop flexibility becomes very practical. Choosing the "right" desktop is not about loyalty to one interface. It is about matching the environment to the task and the hardware.

GNOME is modern and polished, but it can be heavier than some lighter alternatives. Readers working on older or lower-powered systems may find that desktops such as **Xfce**, **Lubuntu**, or other lightweight options feel more responsive and consume fewer resources. This is especially useful for older laptops, virtual machines, training labs, and small devices where every megabyte matters. In contrast, users on newer systems may prefer GNOME or KDE because they offer richer visual integration and more advanced features.

Usability is also personal. Some users enjoy GNOME's streamlined and search-driven workflow. Others prefer a more traditional layout with a visible application menu and a familiar taskbar. Ubuntu's wider flavor ecosystem exists because "best" does not mean the same thing for everyone. A productive desktop is the one that lets the user think clearly, work comfortably, and stay stable on available hardware.

Battery life and responsiveness should also be considered. Heavier desktop environments may provide more visual features, but they can place greater demands on CPU, memory, and power. Lighter desktops often respond faster on low-end hardware and may benefit users who work unplugged for long periods. A wise desktop choice is therefore a balance among **performance**, **comfort**, and **purpose**.

A practical selection mindset is:

- choose **GNOME** for a modern default Ubuntu experience
- choose a **lighter desktop** for older hardware or constrained virtual machines
- choose a **more traditional layout** if user familiarity matters
- test a desktop in real use before deciding permanently

Chapter Reflection

Linux desktop technology may seem complex at first, but its structure is actually elegant. A desktop environment provides the visual workspace. GNOME gives Ubuntu its default modern personality. Wayland and X.Org handle display in different ways, with Wayland leading current Ubuntu use and X.Org remaining valuable for compatibility. Core components such as the display server, window manager, file manager, settings framework, and toolkit all contribute to the user experience. Workspaces and notifications support focused productivity, while Ubuntu's flexibility allows users to choose environments that fit both their habits and their hardware.

Key Takeaways

- **A Linux desktop environment is a complete graphical workspace**, not just a wallpaper and icons.
- **GNOME is Ubuntu Desktop's default environment** and emphasizes clean, focused productivity.
- **Wayland is the modern default in Ubuntu**, while **X.Org** remains useful for compatibility in some cases.
- **Core desktop components** include the display server, window manager, file manager, settings framework, and toolkit.
- **Workspaces and notifications** help organize modern desktop workflows with less clutter.
- **The best desktop choice depends on hardware, workflow, and user preference**, not fashion alone.

Chapter 3 — Touring the Ubuntu Desktop

The first real conversation between a user and Ubuntu Desktop begins not in the Terminal, but on the screen. This chapter introduces the reader to the **visual workflow of Ubuntu Desktop**, especially the default GNOME-based experience used in modern Ubuntu. The goal is not only to help readers recognize the interface, but also to help them **feel at home quickly**. A good desktop should not feel like a maze. It should feel like a workspace that invites exploration, supports concentration, and rewards curiosity. That is exactly why this chapter matters in the Ubuntu Academy learning path: before learners install packages, manage users, or troubleshoot the system, they must first become fluent in moving around the desktop itself.

A useful way to think about Ubuntu Desktop is to imagine entering a modern study room for the first time. The desk is clean. The tools are within reach. The lights, calendar, and quick controls are easy to access. The room is simple on purpose, so that the user can focus on actual work. Ubuntu's desktop follows this same design philosophy. It tries to stay out of the way while still giving fast access to applications, files, settings, and notifications. GNOME is intentionally structured to reduce clutter and help users move from task to task efficiently.

First Login Experience

When a user signs in to Ubuntu Desktop for the first time, the interface appears clean and calm. Instead of filling the screen with menus, icons, and widgets, Ubuntu presents a minimal desktop designed around focus. The top bar remains visible, the Dock sits along the side, and the rest of the screen is left open for applications and work. This design can feel unfamiliar to users coming from other operating systems, but it quickly becomes natural because the most important actions are logically grouped. The top bar provides system status and access points, while the Activities view acts as the central hub for launching and switching tasks.

At first login, the user normally sees:

- the **top bar**
- the **Dock**
- the desktop background
- access to **Activities**
- system menus for **network, sound, power, and session controls**

This first impression matters. Ubuntu Desktop does not try to imitate every other platform. Instead, it introduces a workflow centered on **overview, search, and workspaces**. For students and practitioners, this is a valuable shift. Rather than memorizing many layers of menus, they learn to navigate through **search**, **shortcuts**, and **visual task grouping**. That makes the environment feel modern and efficient rather than crowded.

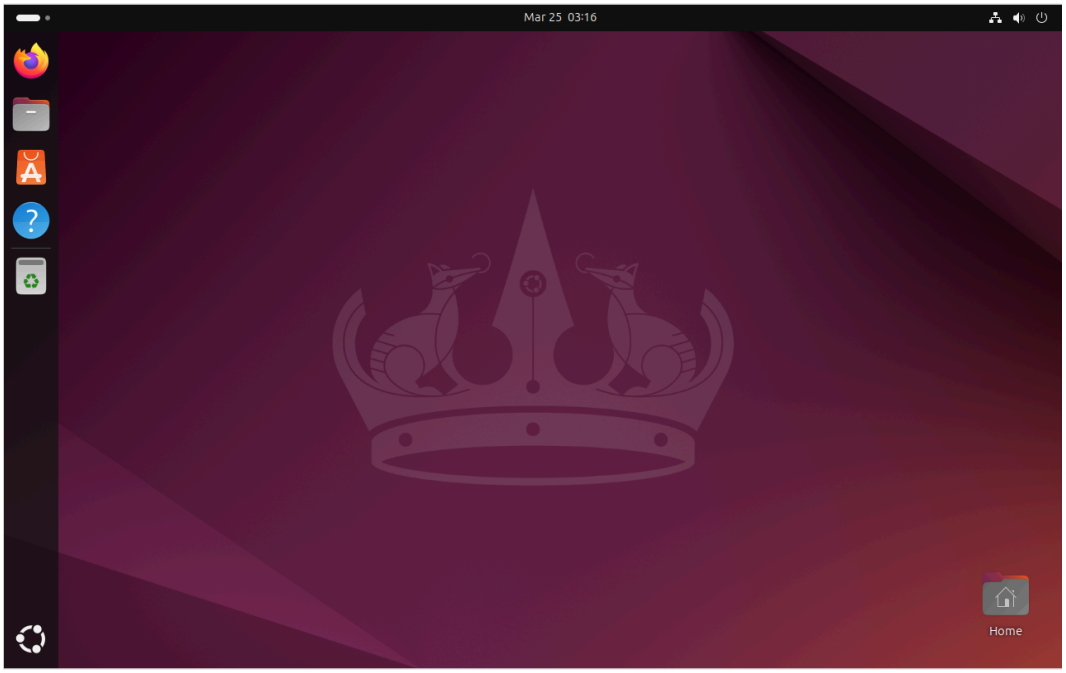


Figure. Ubuntu Desktop

Top Bar, Dock, Activities Overview, and App Grid

The **top bar** is one of the first interface elements users notice. It gives access to essential desktop functions such as the Activities button, clock and calendar, status icons, and the system menu. In daily use, this bar becomes a control strip for checking the time, opening the calendar, seeing battery and network status, adjusting sound, switching users, and powering off the machine. Even though it looks simple, it is one of the busiest and most useful areas of the Ubuntu desktop.

The **Dock** appears on the left side in Ubuntu's default session. It contains favorite applications and also shows running programs. This makes it both a launcher and a visual task switcher. A user can pin commonly used applications there and open them with a single click. The Dock is especially helpful for beginners because it provides visible anchors: browser, files, software tools, terminal, and other daily-use apps remain easy to locate.

The real heart of Ubuntu Desktop, however, is the **Activities Overview**. This is where the desktop becomes dynamic. From the Activities view, users can see open windows, move between workspaces, search for applications and files, and launch new programs. It is not merely a menu screen; it is a central command space for navigating the desktop efficiently. GNOME is strongly designed around this Overview model, which is why pressing the **Super** key is so important in Ubuntu.

Above or within the Overview, the user can access the **App Grid**, which shows installed applications in a more traditional launcher layout. This is useful for learners who want a familiar "show me all my apps" view. In practice, many users gradually move from browsing the grid to simply pressing **Super** and typing the app name, because search is usually faster.

A simple illustration may help:

- The **top bar** is like the control panel above your desk.
- The **Dock** is the row of tools you keep close at hand.
- The **Activities Overview** is the map of your whole workspace.
- The **App Grid** is the full cabinet of available tools.

Screenshots:

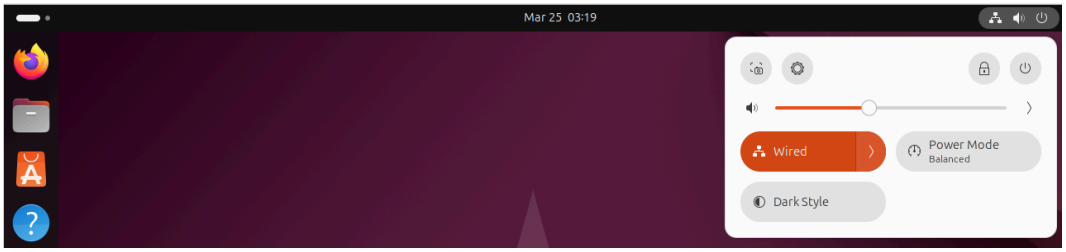


Figure. Desktop Top bar and Dock

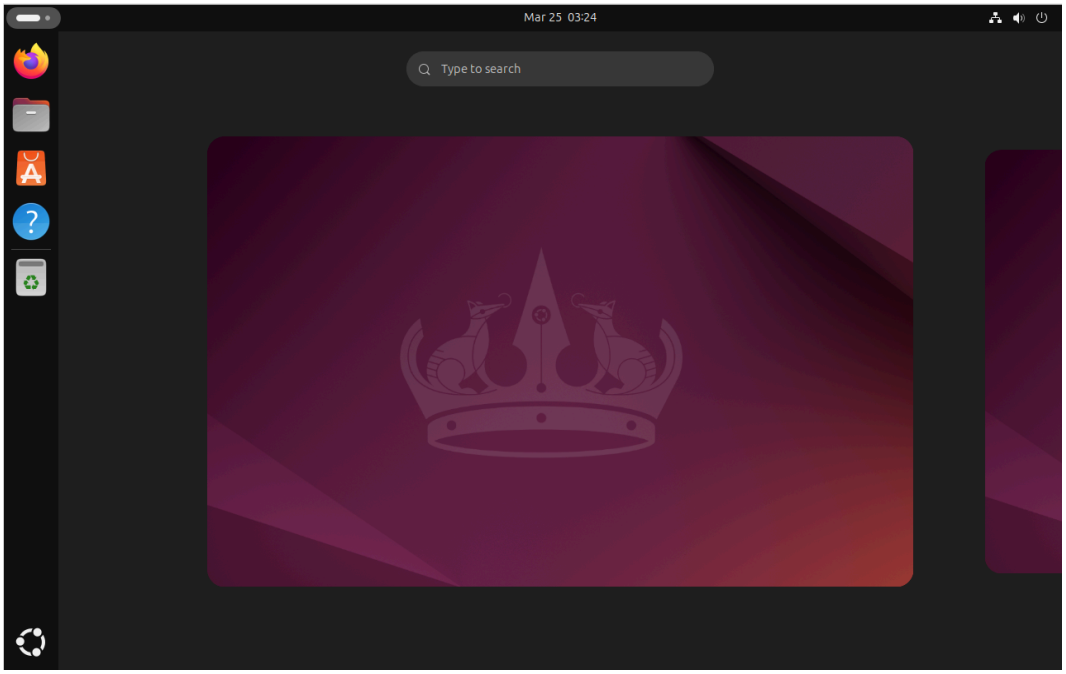


Figure. Activities Overview

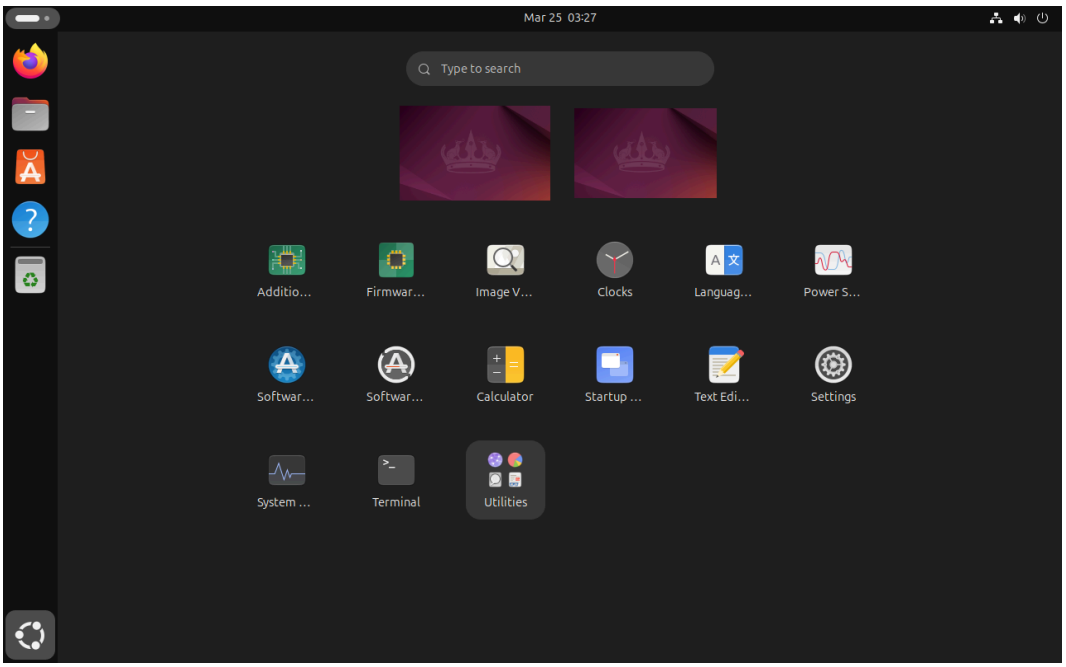


Figure. App Grid Open

Files (Nautilus) and Folder Navigation

For many users, the most important desktop application is the **file manager**. On Ubuntu Desktop, this role is handled by **Files**, historically known as **Nautilus**. It gives users a visual way to browse folders, open documents, move files, access removable drives, and organize their work. A file manager is where the abstract Linux filesystem becomes something practical and understandable. Instead of thinking in terms of paths only, users can navigate through folders such as **Home**, **Documents**, **Downloads**, **Pictures**, and **Desktop** visually.

Files helps readers build an everyday relationship with Ubuntu. They can copy and paste files, drag items between folders, create directories, rename documents, and open external storage devices. These actions may feel ordinary, but they are foundational. A comfortable desktop user is usually a confident file manager user. Learning to understand the **Home folder** is especially important, because it becomes the central personal workspace for each account.

A helpful way to explain this is to compare the Home folder to a student's locker or personal office cabinet. Inside it are organized sections:

- **Documents** for reports and writing
- **Downloads** for files from the internet
- **Pictures** for images and screenshots
- **Music** and **Videos** for media
- hidden configuration files behind the scenes for advanced use

When learners understand this layout, Ubuntu becomes easier and less intimidating.

CLI example: open the Home folder path in Terminal

```
pwd
```

Example output:

```
/home/student
```

This output shows the current working directory. On a normal Ubuntu Desktop account, the user's personal files live under `/home/username`.

CLI example: list common folders in the Home directory

```
ls
```

Example output:

```
Desktop  Documents  Downloads  Music  Pictures
Public  Templates  Videos
```

This is a useful bridge between the graphical Files application and the command line. Readers should learn that the same folders visible in Files can also be inspected from Terminal.

CLI example: create a practice folder

```
mkdir ~/Desktop/ubuntu-tour-lab
ls ~/Desktop
```

Example output:

```
ubuntu-tour-lab
```

This small example helps learners see that desktop actions and terminal actions are closely connected.

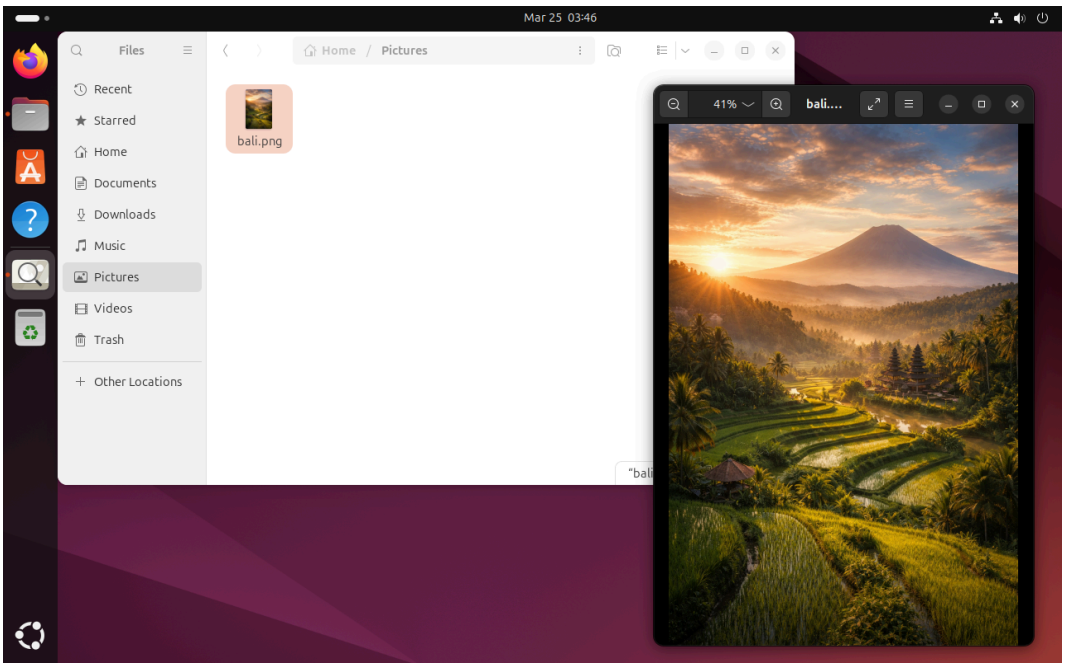


Figure. Navigating Pictures.

Search, Notifications, Calendar, and Quick Settings

One of Ubuntu Desktop's strongest productivity features is **search**. Instead of manually browsing through menus, users can press the **Super** key and start typing. Ubuntu can search for applications, files, settings, and sometimes available software. This search-first workflow is one of the reasons GNOME feels modern and fast. It encourages users to think in terms of intention rather than location: "I want Terminal" becomes **Super**, type `terminal`, press Enter.

The desktop also includes an integrated **calendar and notifications area**. By selecting the clock in the top bar, users can review recent notifications and check the current date. This is useful for desktop awareness without forcing separate applications open all the time. GNOME's design aims to inform the user without turning the desktop into a noisy dashboard. In professional environments, this subtlety is important because concentration matters.

The **Quick Settings** area, usually accessed from the upper-right part of the top bar, provides fast controls for sound, network, brightness, power, and other session options. This is often where users connect to Wi-Fi, check battery status, change audio volume, or prepare to log out. It is one of the most practical parts of the interface because it brings common controls together in a single location.

CLI example: verify current date and time

```
date
```

Example output:

```
Wed Mar 12 10:15:42 WIB 2026
```

This is a simple way to connect the graphical calendar view with a command-line equivalent.

CLI example: check hostname and current user

```
whoami
hostname
```

Example output:

```
student
ubuntu-desktop
```

These commands are useful when readers begin relating the visible desktop session to the identity of the logged-in account and the machine itself.

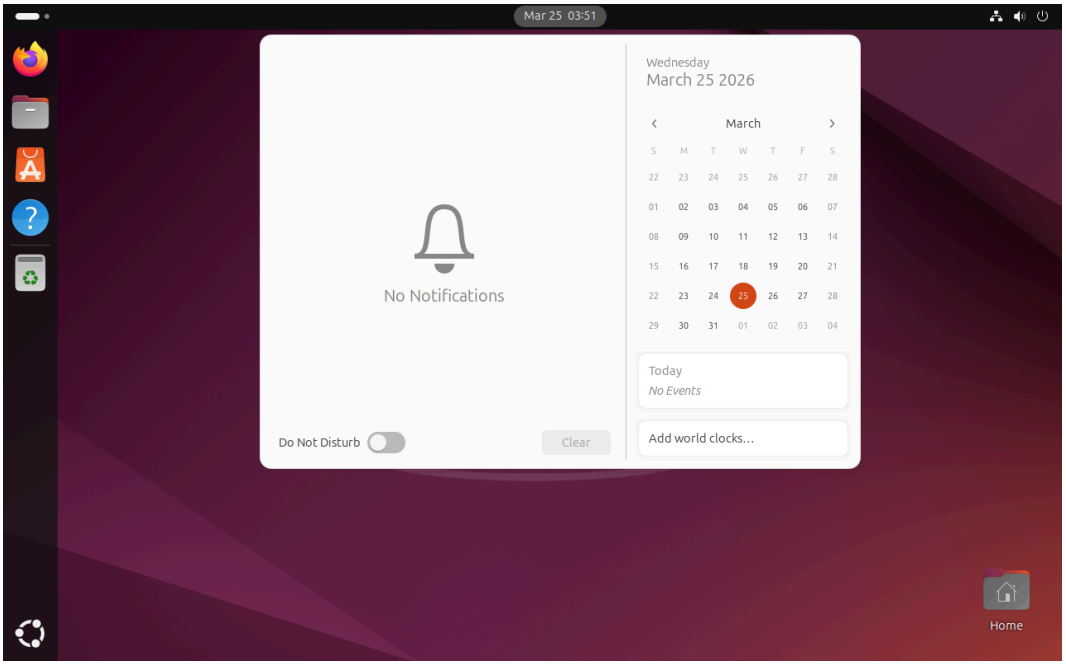


Figure. Calendar and notifications dropdown open

Window Management and Workspaces

An effective desktop is not only about opening applications. It is also about managing them well. Ubuntu Desktop supports **window management** features such as switching between windows, tiling them side by side, maximizing, restoring, and moving them across workspaces. This allows the desktop to scale from simple casual use to serious multitasking.

Workspaces are especially important. They act as virtual desktops, allowing users to separate tasks into different visual areas. For example, a student might keep a web browser and PDF on one workspace, a writer or office document on another, and a chat or file manager window on a third. This reduces clutter and makes context switching easier. GNOME treats workspaces as a normal part of daily productivity, not as an advanced feature reserved for experts.

A simple illustration is to compare one large cluttered desk with several smaller organized desks. Workspaces give the user those smaller desks without needing another monitor. This is especially powerful on laptops, where screen space is limited.

CLI example: open several applications for workspace practice

```
nautilus &
gnome-text-editor &
firefox &
```

Example output:

```
[1] 4121
[2] 4178
[3] 4230
```

The background process numbers are not the important part; the key idea is that multiple applications can be launched and then arranged visually using the desktop.

CLI example: inspect running graphical applications

```
ps -ef | grep -E
'nautilus|firefox|gnome-text-editor' | grep -v grep
```

Example output:

```
student    4121  2980  0 10:22 ?        00:00:01 nautilus
student    4178  2980  0 10:22 ?        00:00:00
gnome-text-editor
student    4230  2980  2 10:22 ?        00:00:03 firefox
```

This example shows that desktop actions are still backed by Linux processes.

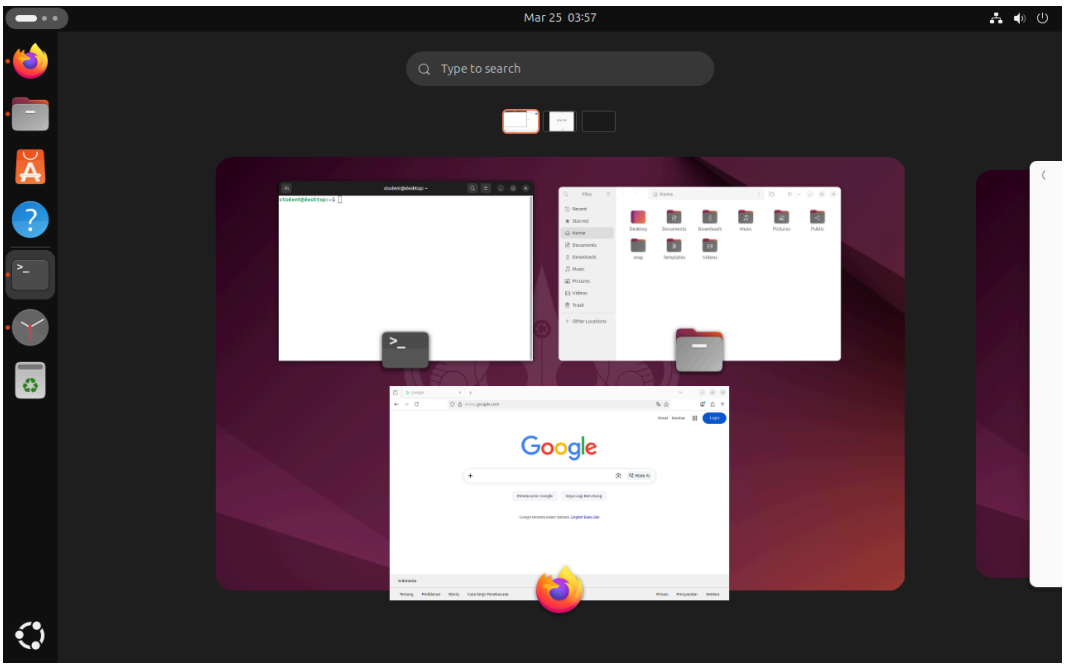


Figure. Task Separation in Ubuntu Desktop.

Accessibility Features

A strong desktop platform should be usable by as many people as possible. Ubuntu Desktop includes **accessibility features** to support users with different visual, auditory, motor, or reading needs. These features are not side issues. They are part of good desktop design. Modern Linux desktops, including GNOME, include accessibility support in both the interface and the toolkit level, helping applications become more usable across different needs.

Depending on the setup, accessibility features may include:

- larger text
- high-contrast display options
- screen reader support
- keyboard navigation support
- visual alerts
- zoom or magnification options
- on-screen keyboard support

For learners and practitioners, the important lesson is that Ubuntu Desktop is not only technically capable; it is also designed to be inclusive. In labs, schools, and public-access computer environments, accessibility is part of professional system preparation.

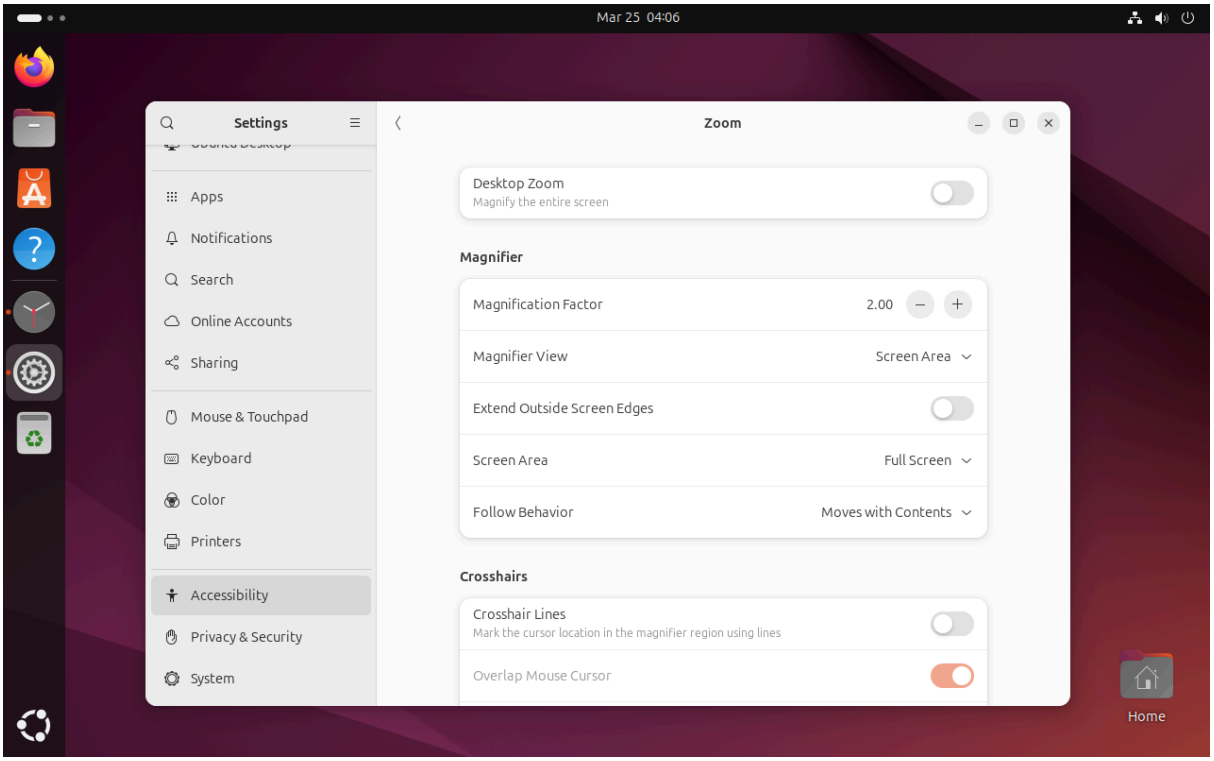


Figure. Accessibility settings showing Magnifier and Crosshairs options.

Power, Session, Lock, Logout, Suspend, Restart, and Shutdown

A desktop session is not only about starting work; it is also about ending or pausing work properly. Ubuntu Desktop provides these controls through the system area in the upper-right section of the top bar. From there, users can access options such as **lock screen**, **log out**, **switch user**, **suspend**, **restart**, and **power off**. These actions are everyday skills, especially on shared systems, lab machines, and laptops.

The distinction between these actions matters:

- **Lock** protects the current session without closing applications.
- **Log Out** ends the user session but leaves the system running.
- **Suspend** saves power while preserving the current session in memory.
- **Restart** reboots the system.
- **Shutdown** powers off the machine completely.

A good illustration is to compare them to leaving a room:

- **Lock** = close the door but keep your desk exactly as it is.
- **Log Out** = clear your desk and leave the room.
- **Suspend** = dim the room and pause activity.
- **Restart** = reopen the room fresh.
- **Shutdown** = close the building for now.

CLI example: lock the current session

```
loginctl lock-session
```

Typical result:

```
(no output if successful)
```

CLI example: reboot or power off from Terminal

```
systemctl reboot
systemctl poweroff
```

These commands should be used carefully, but they help readers see that desktop power actions also have command-line equivalents.

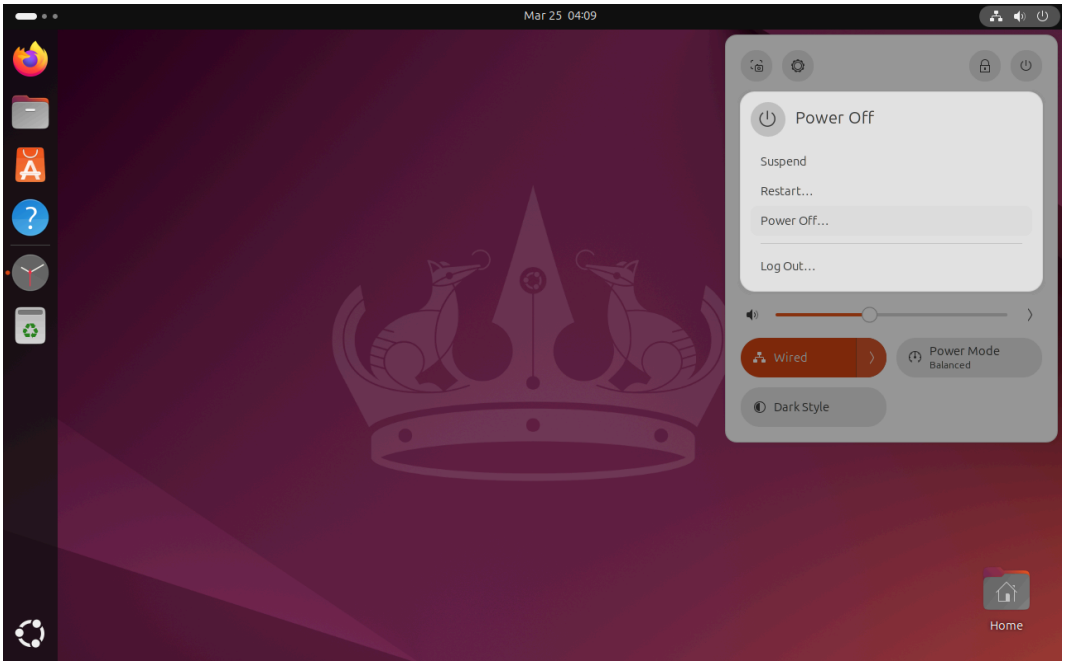


Figure. System menu showing Suspend, Restart, Power off options.

Keyboard Shortcuts for Productive Daily Use

Keyboard shortcuts are one of the fastest ways to become confident on Ubuntu Desktop. They reduce friction, speed up navigation, and make the user feel more in control. Some learners initially avoid shortcuts because they seem advanced, but in reality they are one of the easiest ways to improve desktop fluency. Ubuntu and GNOME provide many helpful shortcuts, and even a small set can dramatically improve productivity.

Some especially useful examples include:

- **Super** → open Activities
- **Alt + Tab** → switch between applications
- **Ctrl + Alt + T** → open Terminal
- **Super + Left / Right** → tile the current window to half the screen
- **Super + Up / Down** → maximize or restore the current window
- **Super + PgUp / PgDn** → move between workspaces
- **Super + Shift + PgUp / PgDn** → move the current window to another workspace

Ubuntu also allows users to explore its keyboard help. Some references note that shortcut discovery is built into the desktop experience, reinforcing the idea that Ubuntu is designed to reward active learning.

CLI example: launch Terminal quickly and inspect the desktop session

After pressing **Ctrl + Alt + T**, users can run:

```
echo $XDG_CURRENT_DESKTOP
echo $DESKTOP_SESSION
```

Example output:

```
ubuntu:GNOME
ubuntu
```

This is a nice teaching moment because it links the visible desktop session to the environment variables behind it.

CLI example: check current graphical session type

```
echo $XDG_SESSION_TYPE
```

Example output:

```
wayland
```

On some systems, the output may instead be:

```
x11
```

This helps learners connect the interface they see with the display technology discussed in the previous chapter.

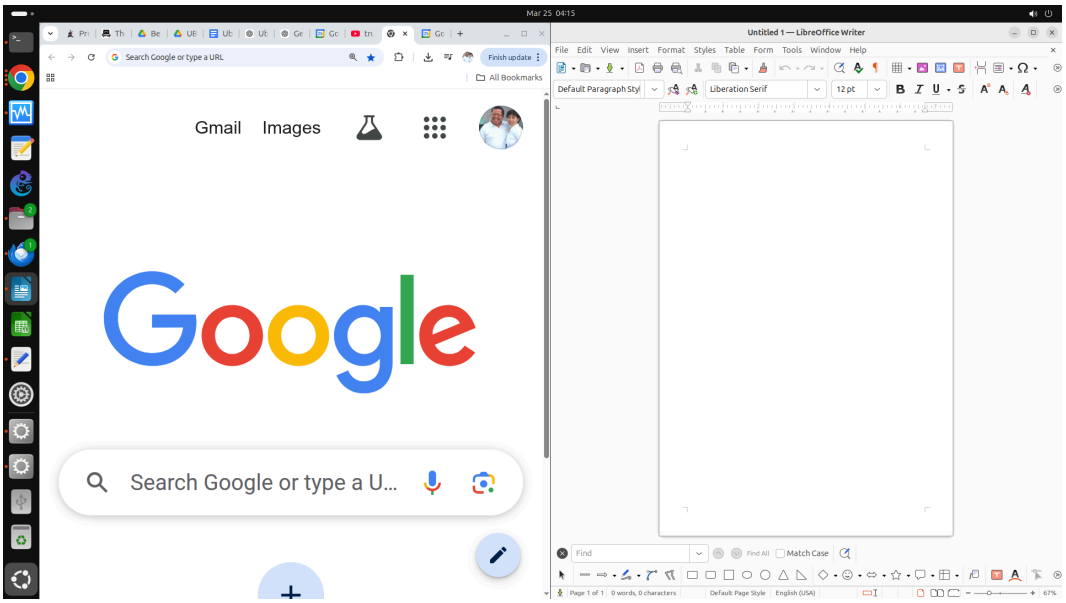


Figure. Super+Left/Right tiling Results.

Guided Desktop Walkthrough Activity

To make this chapter practical and fun, readers can complete a short exploration lab:

1. Log in to Ubuntu Desktop and identify the **top bar**, **Dock**, and **Activities** button.
2. Press **Super** and search for **Files**.
3. Open **Files** and browse to **Documents** and **Pictures**.
4. Open **Terminal** with **Ctrl + Alt + T**.
5. Run `pwd`, `ls`, and `echo $XDG_CURRENT_DESKTOP`.
6. Open a browser and a text editor, then tile them side by side.
7. Move one application to another workspace.
8. Open the calendar and Quick Settings menus.
9. Lock the screen, then sign back in.
10. Practice logging out or suspending the machine safely.

This kind of activity transforms Ubuntu from “something to observe” into **something to use confidently**.

Chapter Reflection

Touring Ubuntu Desktop is more than learning where buttons are located. It is about building a working relationship with the system. The first login introduces a clean and focused environment. The top bar, Dock, Activities Overview, and App Grid provide the main navigation structure. Files help users understand and manage their personal workspace. Search, notifications, calendar, and Quick Settings support a modern workflow. Window management and workspaces make multitasking practical, while accessibility and session controls ensure Ubuntu remains usable and professional in real environments. Keyboard shortcuts then tie everything together, turning simple users into efficient users.

Key Takeaways

- **Ubuntu Desktop is intentionally simple at first glance**, but powerful in daily use.
- **The Activities Overview is the central navigation hub** for launching, switching, and searching.
- **Files (Nautilus) is essential for understanding the user’s workspace and folders.**
- **Search and shortcuts are often faster than menu browsing.**
- **Workspaces help reduce clutter and improve focus.**
- **Quick Settings and session controls are core daily-use tools.**
- **A small set of keyboard shortcuts can greatly improve productivity.**

Part II — Installation and First-Time Setup

Chapter 4 — Preparing and Installing Ubuntu Desktop

Installing Ubuntu Desktop is one of the most exciting stages in the learning journey because it turns a curious reader into an actual Ubuntu user. This chapter is designed to help students and practitioners prepare properly, choose the right installation approach, and understand what the installer is doing without making the process feel intimidating. In the Ubuntu Academy context, this chapter supports the **Installing Ubuntu Desktop** domain, including boot media creation, interactive installation choices, encryption, and first-boot preparation. Ubuntu's desktop-focused install experience is intentionally designed to be clear, guided, and approachable, especially compared with older Linux installation methods.

A useful mindset for this chapter is to think of installation as preparing a new workspace. Before moving into a new office or study room, you check whether the room is large enough, whether the electricity works, whether the desk fits your needs, and whether you want to share the space with someone else. Installing Ubuntu is very similar. You check hardware, decide which edition to use, prepare installation media, choose where Ubuntu will live, and then configure the system so it is ready for productive daily work.

Hardware Requirements and Compatibility Checks

Before installing Ubuntu Desktop, the first step is to make sure the machine is suitable for a modern desktop environment. Ubuntu Desktop is not extremely demanding, but it still needs a reasonable processor, sufficient memory, storage space, and the ability to boot from installation media. Recent Ubuntu guidance in the reference material points to a **2 GHz-class processor**, **at least 4 GB of RAM for desktop use**, and around **25 GB of disk space** as a comfortable starting point for a normal desktop installation. Older beginner-oriented material also frames Ubuntu Desktop as requiring more resources than Ubuntu Server because Desktop includes a full graphical environment and daily-use applications.

Compatibility is not only about raw specifications. It is also about whether important hardware is recognized properly. In desktop scenarios, readers should care about:

- **graphics support**
- **Wi-Fi chipset support**
- **Bluetooth support**
- **touchpad and keyboard behavior**
- **sound devices**
- **UEFI/boot behavior**
- external display support if using laptops or docking stations

Ubuntu Live Mode is especially valuable here because it allows a user to test major hardware functions before committing to a full install. If Wi-Fi, display brightness,

keyboard layout, and sound work correctly in the live session, that is a very encouraging sign for the real installation. The references also note that if a computer is less powerful than recommended, it may be better to consider a lighter Ubuntu-based desktop environment instead of forcing the default GNOME desktop onto unsuitable hardware.

CLI example: inspect hardware from a live session

```
lscpu
free -h
lsblk
```

Example output:

```
Architecture:        x86_64
CPU(s):              4
Model name:          Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

               total        used        free      shared  buff/cache   available
Mem:           7.6Gi       1.4Gi       4.8Gi       210Mi       1.4Gi       5.8Gi
Swap:          2.0Gi          0B       2.0Gi

NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 238.5G  0 disk
├─sda1   8:1    0   512M  0 part
├─sda2   8:2    0   100G  0 part
└─sda3   8:3    0   138G  0 part
sr0     11:0    1   4.7G  0 rom
```

This gives a quick overview of processor, memory, and storage layout.

CLI example: detect hardware devices

```
sudo lshw -short
```

Example output:

```
H/W path        Device      Class          Description
=====================================================
/0                          system         Laptop
/0/0                        memory         8GiB System Memory
/0/1                        processor      Intel(R) Core(TM) i5-8250U CPU
/0/100/1f.3                 multimedia     Audio device
/0/100/14.3                 network        Wireless interface
/0/100/2                    display        VGA compatible controller
```

This is a practical compatibility check before installation.

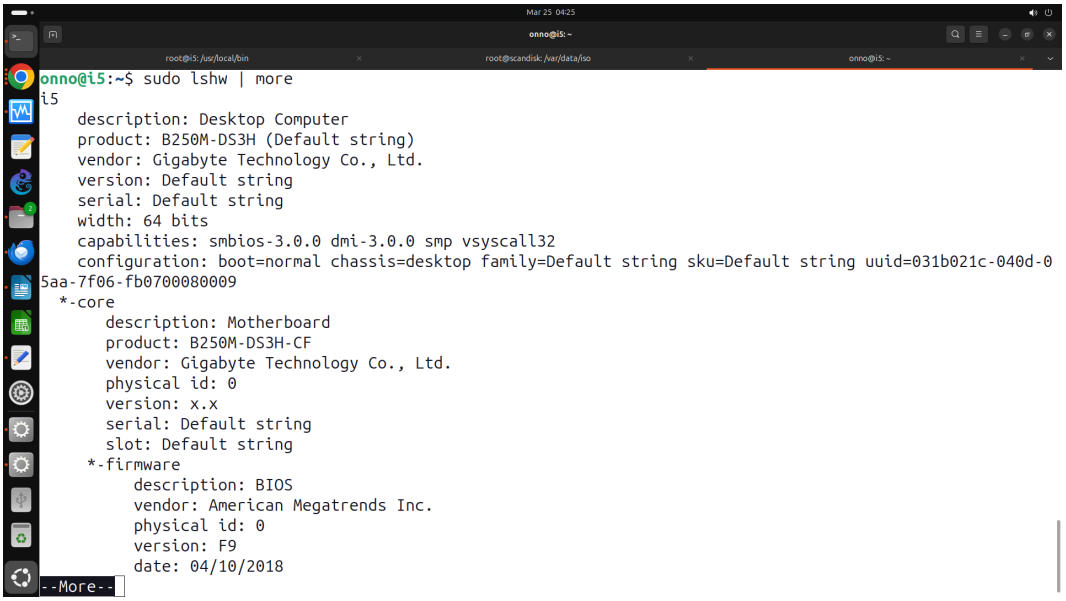


Figure. lshw output

Choosing the Right Ubuntu Desktop Edition

Choosing the right Ubuntu Desktop edition is not only about downloading “Ubuntu.” It is about selecting the version that best fits the user’s purpose, hardware, and learning goals. The Ubuntu Academy outline is centered on the standard **Ubuntu Desktop** experience with GNOME, and that should remain the primary focus for this book. However, the references also remind readers that Ubuntu exists in multiple desktop forms and related flavors, which can be useful when hardware or workflow needs differ.

For most readers of this book, the best recommendation is:

- choose **Ubuntu Desktop LTS**
- use the **default GNOME-based edition**
- prefer **64-bit** installation media on modern systems
- choose lighter alternatives only when hardware limitations clearly justify it

An LTS release is usually the safest option for learning, labs, institutions, and day-to-day work because it prioritizes stability and long-term support. A student who is still learning the desktop should not be distracted by unnecessary upgrade pressure. A stable platform makes it easier to focus on skills. The references also show that Ubuntu’s success comes partly from adding a simpler graphical installer and a stronger desktop focus on top of Debian’s software base, which is why Ubuntu Desktop remains such a strong educational choice.

A helpful illustration is this: selecting the edition is like choosing the right classroom. You do not need the fanciest room. You need the one where the chair is comfortable, the lights work, and the tools match the lesson.

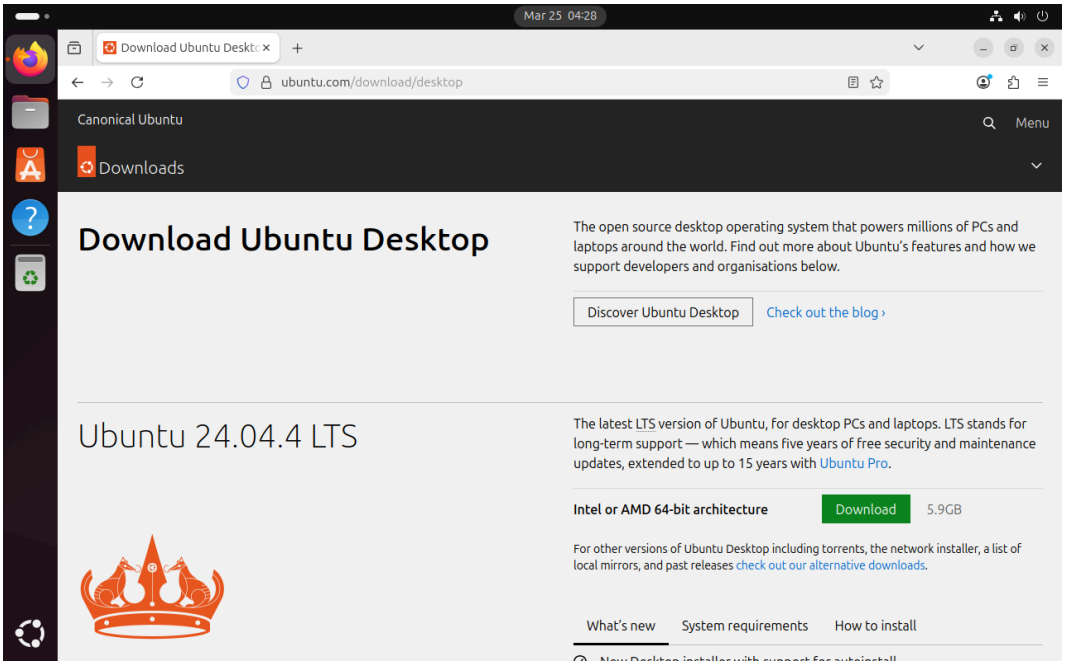


Figure. Ubuntu Desktop LTS download page.

Creating Bootable Installation Media

Once the correct ISO image has been downloaded, the next task is to turn that image into bootable installation media. The most common choice is a USB flash drive. In the Ubuntu Academy outline, three methods are explicitly emphasized: **Rufus**, **Startup Disk Creator**, and **dd**. These methods represent three different learning styles: Windows-friendly GUI, Ubuntu-friendly GUI, and command-line precision.

A bootable USB is like a portable installer workshop. Instead of installing Ubuntu from inside the existing operating system in the usual way, the computer boots directly into the installer environment from the USB drive. From there, the user can try Ubuntu in live mode or install it permanently. The reference material also notes that many systems today use UEFI-based booting, so users may need to check their firmware boot menu or startup key sequence in advance.

Before creating the media, readers should remember:

- back up anything important on the USB drive
- use a reputable ISO source
- verify that the USB drive is large enough
- make sure the target machine is able to boot from USB
- safely eject the USB after writing is complete

Rufus

Rufus is one of the easiest tools for Windows users who want to create a bootable Ubuntu USB. It provides a simple graphical workflow: choose the ISO file, select the USB drive, review the partition scheme and target system if necessary, and start the write process. For many learners coming from Windows, Rufus is the most comfortable entry point because it minimizes command-line complexity and gives visual feedback. The beginner-oriented Ubuntu reference explicitly discusses creating bootable media from other operating systems, including Windows-based workflows.

Typical Rufus workflow:

- insert the USB drive
- launch Rufus
- select the Ubuntu ISO
- confirm the target USB device
- start writing
- wait for completion
- reboot and boot from USB

Startup Disk Creator

On Ubuntu itself, **Startup Disk Creator** offers a straightforward graphical method for writing Ubuntu images to USB media. This is especially useful when one Ubuntu machine is used to prepare another. The beginner reference explicitly includes steps for creating a bootable USB using Ubuntu's own tools, which is valuable for lab environments and teaching scenarios where one Linux workstation prepares multiple installers.

Typical Startup Disk Creator workflow:

- plug in the USB drive
- open **Startup Disk Creator**
- select the downloaded ISO image
- select the destination USB device
- write the image
- wait for completion

This method is excellent for students because it keeps the process visual and safe.

dd

The command-line tool `dd` is more advanced, but it is also one of the most important Linux tools for writing raw disk images. It is powerful and fast, but it must be used carefully because writing to the wrong device can destroy data. The reference material mentions `dd` directly in the Ubuntu Academy outline and also notes its use from live or recovery environments for disk-level operations.

CLI example: identify the USB device before writing

```
lsblk
```

Example output:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 238.5G  0 disk
├─sda1   8:1    0   512M  0 part /boot/efi
└─sda2   8:2    0   238G  0 part /
sdb      8:16   1  15.0G  0 disk
└─sdb1   8:17   1  15.0G  0 part /media/ubuntu/USB
```

Here, `/dev/sdb` is the USB device.

CLI example: unmount the USB before writing

```
sudo umount /dev/sdb1
```

Typical result:

```
(no output if successful)
```

CLI example: write the Ubuntu ISO to USB with dd

```
sudo dd if=ubuntu-24.04-desktop-amd64.iso
of=/dev/sdb bs=4M status=progress oflag=sync
```

Example output:

```
2147483648 bytes (2.1 GB, 2.0 GiB) copied, 48 s,
44.7 MB/s
512+1 records in
512+1 records out
2149580800 bytes (2.1 GB, 2.0 GiB) copied, 49.2103
s, 43.7 MB/s
```

CLI example: flush remaining writes

```
sync
```

This ensures data is fully written before removing the USB drive.

A simple warning readers should remember: **`dd` does exactly what you tell it to do, not what you meant to do**. Always verify the target device first.

Trying Ubuntu in Live Mode

One of Ubuntu Desktop's most beginner-friendly strengths is **Live Mode**. Instead of forcing the user to install immediately, Ubuntu can boot into a temporary desktop session directly from the USB drive. This session allows the user to explore the desktop, connect to Wi-Fi, test sound, browse files, and confirm hardware compatibility before touching the internal disk. The references explicitly describe this "try before install" experience and note that the live system runs more slowly than a full installation because it is operating from removable media.

Live Mode is like test-driving a car before buying it. The goal is not to do all your long-term work there. The goal is to see whether the environment feels right and whether the essential systems behave correctly.

In Live Mode, readers should test:

- display resolution and scaling
- Wi-Fi connection
- keyboard and touchpad
- audio playback
- Bluetooth if needed
- file manager and external USB detection
- overall desktop responsiveness

CLI example: confirm the session type in live mode

```
echo $XDG_SESSION_TYPE
echo $XDG_CURRENT_DESKTOP
```

Example output:

```
wayland
ubuntu:GNOME
```

CLI example: check network connection

```
ip a
ping -c 3 8.8.8.8
```

Example output:

```
2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.1.23/24 brd 192.168.1.255 scope global
dynamic wlp2s0

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=22.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=23.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=22.8 ms
```

This gives quick confirmation that networking works before installation.

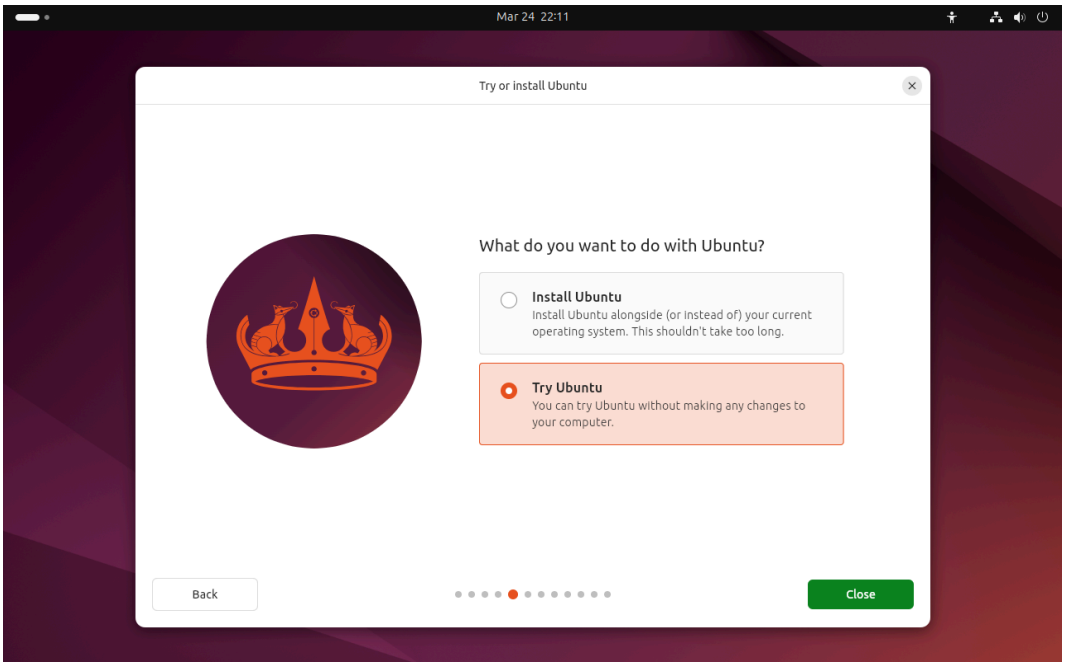


Figure. Try Ubuntu Option.

Clean Install, Dual Boot, and VM Installation

Ubuntu Desktop can be installed in several ways, and each one fits a different learning or deployment scenario.

A **clean install** means Ubuntu takes over the target system or selected target disk space as the main operating system. This is often the simplest and most stable route for dedicated Ubuntu systems. It is ideal for lab PCs, old laptops being repurposed, or users committed to Ubuntu as their main platform.

A **dual boot** setup allows Ubuntu to coexist with another operating system, commonly Windows. The reference material explains dual booting as sharing a disk between Windows and Linux partitions and choosing which operating system to boot at startup. This is very useful for learners who still need access to Windows-specific applications while building Linux skills. It is powerful, but it requires more care with partitions, boot behavior, and backups.

A **virtual machine installation** places Ubuntu inside software such as VirtualBox. This is one of the safest ways to learn because it avoids changes to the host machine's real disk layout. It is especially useful in classrooms, tutorials, and experimentation. The Ubuntu Linux Bible exercises explicitly reference Ubuntu installation through VirtualBox as a practical learning path.

A practical comparison:

- **Clean install** → best performance, full commitment
- **Dual boot** → best when two real operating systems are both needed
- **VM install** → safest for learning, testing, and repeatable labs

CLI example: check current partitions before planning a dual boot

```
sudo fdisk -l
```

Example output:

```
Disk /dev/sda: 238.47 GiB, 256060514304 bytes, 500118192 sectors
Device       Start       End   Sectors   Size Type
/dev/sda1     2048   1050623   1048576   512M EFI System
/dev/sda2  1050624 210000000 208949377  99.7G Microsoft basic data
/dev/sda3 210000001 500118158 290118158 138.4G Linux filesystem
```

This kind of output helps readers visualize whether free space or an existing partition layout is available.

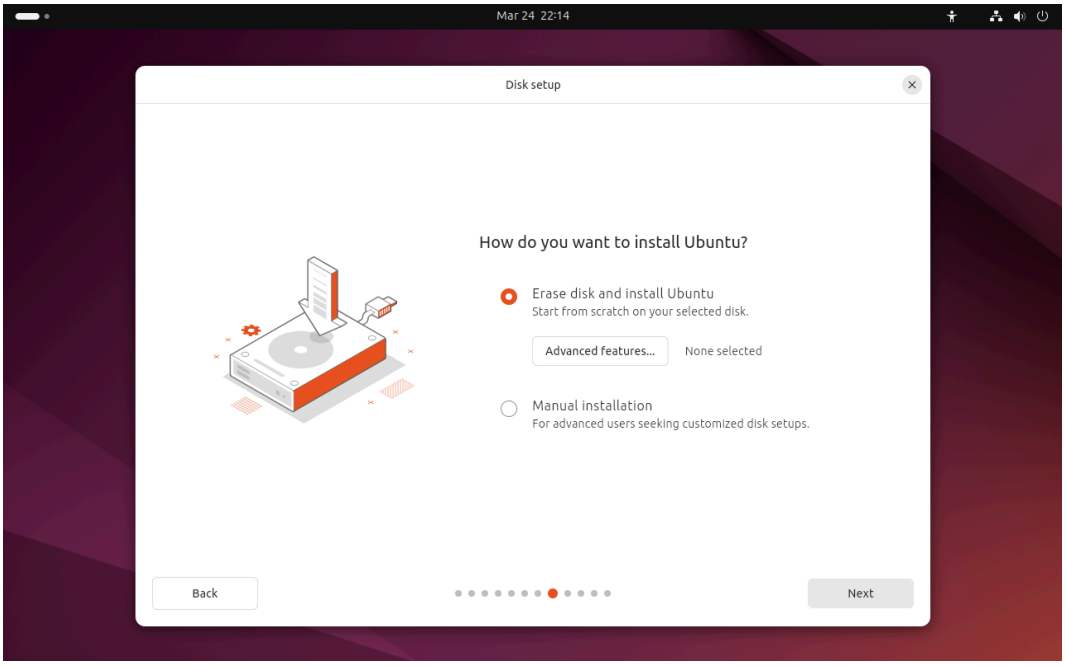


Figure. Ubuntu Installation options.

Installer Options

Ubuntu's installer is designed to guide users through key decisions without overwhelming them. Still, those choices matter. Readers should understand what they are selecting, not just click through.

Minimal vs full install

A **minimal install** provides a leaner starting point with fewer preinstalled applications. This can be useful for low-storage systems, virtual machines, or users who prefer to build their software environment gradually. A **full install** includes a broader set of desktop applications and better supports immediate productivity. The Ubuntu Linux Bible specifically notes that the installer asks how much software should be included in the initial installation, making this a clear early decision point.

A helpful way to explain this is:

- **Minimal install** = moving into an apartment with only the essential furniture
- **Full install** = moving in with the common daily-use furniture already provided

For most beginners, **full install** is often the more comfortable starting point unless storage is tight.

Third-party drivers and codecs

The installer may offer an option to include **third-party software**, such as proprietary drivers and multimedia codecs. This can be important for NVIDIA graphics, certain Wi-Fi chipsets, and media playback formats. The Ubuntu Linux Bible specifically notes that Ubuntu makes it easy to select additional codecs during the standard installation process. In desktop learning environments, enabling this option is often practical because it improves hardware support and day-one usability.

That said, readers should understand why this option exists: not all software used in real-world desktop computing is distributed under the same free-software terms as the core system. Ubuntu gives the user a choice.

Disk partitioning basics

Disk partitioning is where installation becomes more serious. A partition is a defined section of a disk reserved for a particular purpose. The beginner reference explains that in dual-boot scenarios, Windows and Linux typically occupy different partitions and a boot menu lets the user choose between them at startup. It also warns that partitioning is one of the most sensitive parts of installation and that important files should be backed up first.

For desktop learners, the first principle is simple: **do not manually repartition unless you understand the consequences**. Automatic guided options are usually safer for beginners. Still, readers should know the basic concepts:

- **EFI System Partition** for booting on UEFI systems
- root partition `/` for the operating system
- optional separate `/home` for user data
- swap as memory support, depending on system design

The Ubuntu Linux Bible's installation chapters also discuss partitioning and assign partitions to directories in more advanced scenarios, reinforcing that partition design becomes more important in advanced administration.

Encryption options

Ubuntu Desktop also allows **disk encryption**, commonly full-disk encryption using LUKS-based mechanisms in desktop contexts. In the Ubuntu Academy outline, encryption appears as an important desktop security topic. Enabling encryption protects data at rest, which is especially valuable for laptops, portable systems, or machines containing sensitive information.

Encryption is like locking all of the filing cabinets in your office, not just closing the door. Even if the machine is stolen, the data is much harder to access without the passphrase.

Readers should understand the trade-off:

- **better protection for lost or stolen devices**
- slightly more setup responsibility because the passphrase becomes critical

First Boot and Initial Configuration

After installation completes and the system reboots into the newly installed environment, the first boot is where Ubuntu begins to feel personal. The references describe this stage as including post-install setup and initial updates, and the Ubuntu Linux Bible notes that the installer gathers account and location details as part of the process. The purpose of first boot is not only to confirm that installation succeeded, but to make the system ready for real use.

At first boot, readers should verify:

- the user account signs in correctly
- network connectivity works
- display resolution is correct
- sound works
- the clock, region, and keyboard layout are correct
- updates are available and can be installed
- if needed, proprietary drivers can now be added later through the desktop tools mentioned in the Ubuntu Academy outline

This is also the moment to establish good habits. A newly installed system should be updated early, tested calmly, and documented if part of a classroom or lab rollout.

CLI example: refresh package metadata after first boot

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
Building dependency tree... Done
```

All packages are up to date.

CLI example: install available updates

```
sudo apt upgrade -y
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Calculating upgrade... Done
The following packages will be upgraded:
  firefox linux-firmware
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 98.3 MB of archives.
After this operation, 12.4 MB of additional disk space will be used.
```

CLI example: check which desktop session is running

```
echo $XDG_CURRENT_DESKTOP
echo $XDG_SESSION_TYPE
```

Example output:

```
ubuntu:GNOME
wayland
```

CLI example: see which packages can still be upgraded

```
apt list --upgradable
```

Example output:

```
Listing... Done
firefox/noble-updates 1:1snap1-0ubuntu2 amd64
[upgradable from: 1:1snap1-0ubuntu1]
```

These are simple but powerful post-install checks aligned with the practical desktop-admin direction of the Ubuntu Academy outline.

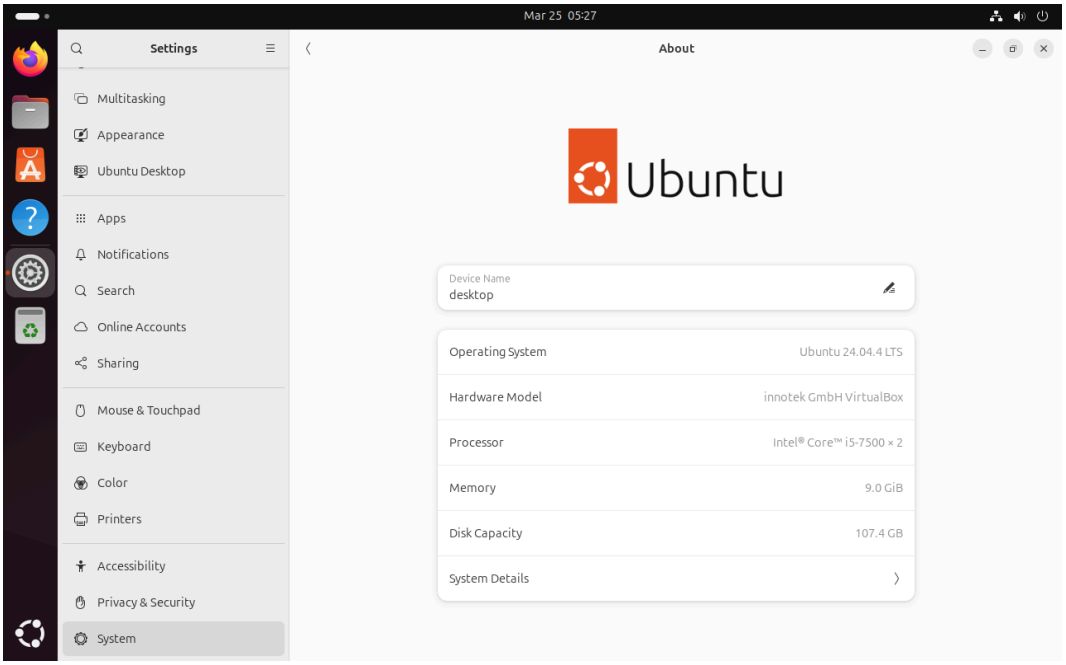


Figure. Setting > About Screen showing Ubuntu version.

Guided Installation Practice

To make this chapter hands-on, readers can follow this short activity:

- download the correct **Ubuntu Desktop LTS** ISO
- create a bootable USB with **Rufus**, **Startup Disk Creator**, or **dd**
- boot into **Live Mode**
- test Wi-Fi, sound, display, and keyboard
- choose one installation path: **VM**, **clean install**, or **dual boot**
- review installer options carefully
- decide between **minimal** or **full** install
- enable **third-party drivers/codecs** if appropriate
- complete the install
- after first boot, run `sudo apt update && sudo apt upgrade -y`

This sequence turns installation from a risky mystery into a guided technical skill.

Chapter Reflection

Preparing and installing Ubuntu Desktop is not just about clicking **Install**. It is about making sound decisions before, during, and after the process. Readers begin by checking whether the hardware is suitable and whether Ubuntu Live Mode works well on the target machine. They then choose the right Ubuntu Desktop edition, prepare a bootable USB, and decide whether to install Ubuntu as the main OS, alongside another system, or inside a virtual machine. During installation, they must understand choices such as minimal versus full installation, third-party software support, partitioning, and encryption. After the first boot, they verify functionality, update the system, and begin shaping Ubuntu into a reliable daily-use desktop.

Key Takeaways

- **Good installation starts with good preparation**, especially hardware and compatibility checks.
- **Ubuntu Desktop LTS** is usually the best choice for learners and stable daily work.
- **Rufus, Startup Disk Creator, and `dd`** are three practical ways to create bootable installation media.
- **Live Mode** is one of the safest ways to test Ubuntu before installation.
- **Clean install, dual boot, and VM installation** each serve different learning and deployment needs.
- **Partitioning and encryption decisions matter**, especially on shared or portable systems.
- **The first boot is part of the installation process**, because updates and verification complete the setup.

Chapter 5 — Post-Installation Essentials

A successful Ubuntu installation is only the beginning. The real desktop experience starts **after the first login**, when the system must be connected, updated, adjusted, and equipped for daily work. In the Ubuntu Academy outline, this stage is directly tied to **post-install setup**, **initial updates**, **hardware support**, and practical desktop readiness. That makes this chapter especially important for students, lab users, office workers, and anyone who wants Ubuntu Desktop to move quickly from “fresh install” to “real workstation.”

A helpful illustration is to imagine moving into a new apartment. The room already exists, but it is not fully ready until the electricity works, the internet is connected, the language settings feel natural, and the tools you use every day are in place. Ubuntu Desktop is similar. The operating system may already be installed, but it becomes truly useful only when the network works properly, updates are applied, drivers are correct, and daily applications are ready.

Connecting to the Internet

One of the first post-installation tasks is connecting Ubuntu Desktop to the internet. Ubuntu’s desktop workflow is strongly designed around online access because software repositories, updates, additional language support, and many application sources are reached through the network. The Ubuntu Linux Bible notes that once Ubuntu is installed, it is automatically prepared to connect to Debian-based software repositories over the internet, allowing users to install and update software through the desktop tools or package manager.

For most users, this means using **NetworkManager** through the graphical desktop. On a laptop, the user usually selects the Wi-Fi icon, chooses a network, enters the password, and waits for the connection to establish. On a wired desktop, the connection may come up automatically if the Ethernet device is recognized. The Ubuntu Academy outline also highlights **Wi-Fi/Ethernet setup**, **VPN**, and **troubleshooting connectivity** as practical desktop skills, so networking is not a side topic. It is part of the system’s day-one usability.

A simple rule for new users is this: before installing extra software or troubleshooting anything complex, confirm that the network is healthy. A surprising number of post-install problems are really connection problems.

CLI example: verify network interface status

```
ip a
```

Example output:

```
2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.1.24/24 brd 192.168.1.255 scope global dynamic wlp2s0
3: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500
```

This output shows that the wireless interface has an IP address and is currently connected.

CLI example: test internet access

```
ping -c 3 8.8.8.8
ping -c 3 archive.ubuntu.com
```

Example output:

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=23.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=22.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=23.5 ms

PING archive.ubuntu.com (185.125.190.82) 56(84) bytes of data.
64 bytes from archive.ubuntu.com: icmp_seq=1 ttl=52 time=189 ms
64 bytes from archive.ubuntu.com: icmp_seq=2 ttl=52 time=191 ms
64 bytes from archive.ubuntu.com: icmp_seq=3 ttl=52 time=188 ms
```

The first test checks raw connectivity. The second confirms that DNS resolution also works.

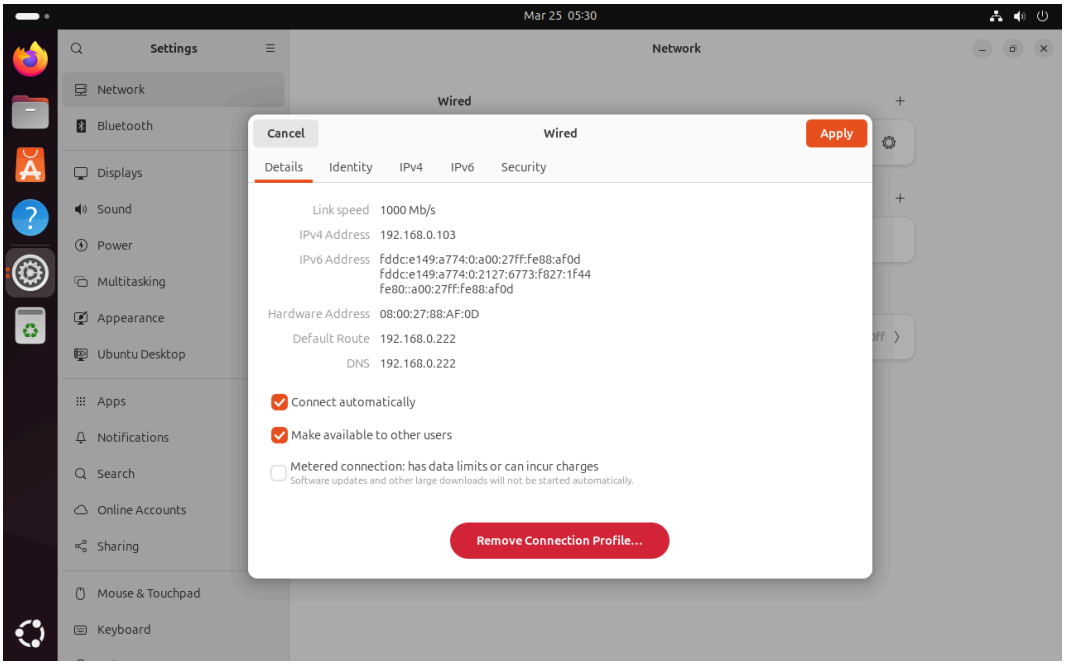


Figure. Settings > Network.

Updating the System After Install

After the network is working, the next essential step is updating the system. This is one of the healthiest habits a Ubuntu user can develop. The Ubuntu Academy outline explicitly includes **post-install initial updates**, **manual update workflows**, and **security patching**, while the reference material repeatedly emphasizes that Ubuntu connects to its repositories so software can be updated and maintained easily.

Updating after installation matters for several reasons:

- it brings in the latest **security fixes**
- it improves **hardware support**
- it refreshes package metadata
- it often resolves early bugs or missing desktop polish
- it prepares the system for later app installation and driver work

For many readers, the easiest path is the graphical **Software Updater** or App Center update flow. However, Ubuntu Desktop users should also become comfortable with the command line, because it is direct, transparent, and excellent for learning.

CLI example: refresh package metadata

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
Building dependency tree... Done
All packages are up to date.
```

CLI example: install available package updates

```
sudo apt upgrade -y
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Calculating upgrade... Done
The following packages will be upgraded:
  firefox linux-firmware
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 97.4 MB of archives.
After this operation, 11.8 MB of additional disk space will be used.
```

CLI example: list packages that can still be upgraded

```
apt list --upgradable
```

Example output:

```
Listing... Done
firefox/noble-updates 1:1snap1-0ubuntu2 amd64 [upgradable
from: 1:1snap1-0ubuntu1]
```

This update workflow aligns directly with the Ubuntu Academy focus on APT basics and practical desktop administration.

A good analogy is this: updates are like bringing a new textbook edition up to date with errata and corrections before the semester begins. The system may already work, but it works **better and safer** after updating.

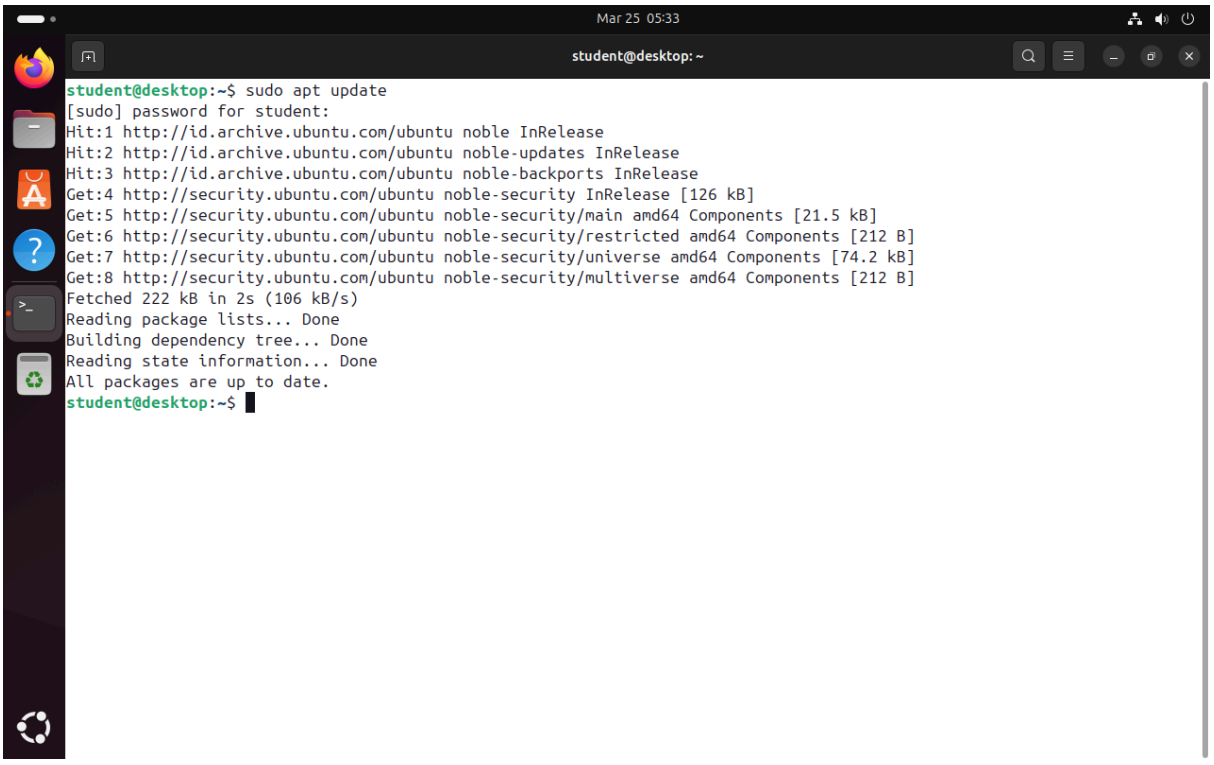


Figure. Update repository.

Installing Proprietary Drivers

Ubuntu usually installs and uses open-source drivers automatically, and for many users those drivers are already enough for normal desktop work. The reference material explains that Ubuntu often ships with open drivers that “just work” for graphics, daily browsing, document work, and general use. But it also points out that users with newer or more demanding hardware may benefit from manufacturer-provided proprietary drivers, especially for graphics and certain wireless chipsets. The Ubuntu Academy outline explicitly includes **installing proprietary drivers** for **NVIDIA/AMD GPUs** and **Broadcom/Realtek Wi-Fi**.

A useful principle for readers is this: **do not install proprietary drivers just because they exist**. Install them when there is a real benefit, such as better performance, improved compatibility, or missing hardware support.

Ubuntu Desktop commonly exposes driver choices through **Software & Updates** and the **Additional Drivers** tab. The reference material specifically describes this workflow for graphics hardware, where compatible proprietary drivers can be selected and applied through the desktop GUI.

CLI example: identify graphics hardware

```
lspci | grep -i vga
```

Example output:

```
00:02.0 VGA compatible controller: Intel Corporation UHD Graphics 620
01:00.0 VGA compatible controller: NVIDIA Corporation GP108M [GeForce MX150]
```

This is a fast way to determine what kind of graphics hardware is present before making driver decisions.

CLI example: ask Ubuntu which drivers are available

```
sudo ubuntu-drivers devices
```

Example output:

```
== /sys/devices/pci0000:00/0000:00:02.0 ==
modalias : pci:v00008086d00005917sv00001028sd00000811bc03sc00i00
vendor   : Intel Corporation
model    : UHD Graphics 620

== /sys/devices/pci0000:00/0000:01:00.0 ==
modalias : pci:v000010DEd00001D10sv00001028sd00000811bc03sc00i00
vendor   : NVIDIA Corporation
model    : GP108M [GeForce MX150]
driver   : nvidia-driver-535 - third-party non-free recommended
driver   : xserver-xorg-video-nouveau - distro free builtin
```

This output helps users compare the open driver with the recommended proprietary one

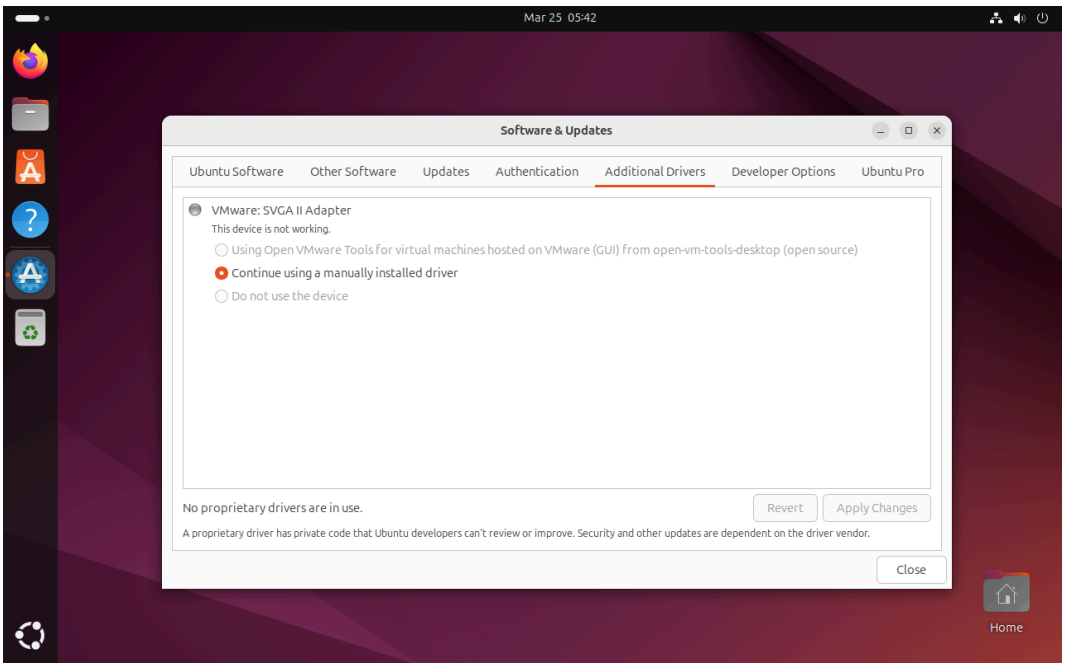


Figure. Additional Drivers in Software & Updates.

NVIDIA

For NVIDIA graphics cards, Ubuntu often works out of the box using the open-source **Nouveau** driver, but the reference material explains that users who want stronger performance, especially for gaming or heavier graphics workloads, may prefer NVIDIA's proprietary driver. It also notes that Ubuntu can expose supported proprietary drivers through the **Additional Drivers** tab, where the recommended tested version can be selected and applied.

This creates a very practical decision:

- use **Nouveau** for simplicity and everyday reliability
- use the proprietary **NVIDIA driver** when performance or compatibility requires it

CLI example: install recommended NVIDIA driver

```
sudo ubuntu-drivers autoinstall
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  nvidia-driver-535 nvidia-dkms-535 libnvidia-gl-535
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Done.
```

After installation, a reboot is usually required.

CLI example: verify loaded NVIDIA driver

```
nvidia-smi
```

Example output:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.171.04    Driver Version: 535.171.04    CUDA Version: 12.2   |
+-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
|  0   GeForce MX150      Off   | 00000000:01:00.0 Off |                  N/A |
+-------------------------------+----------------------+----------------------+
```

This is a reassuring sign that the proprietary driver is active.

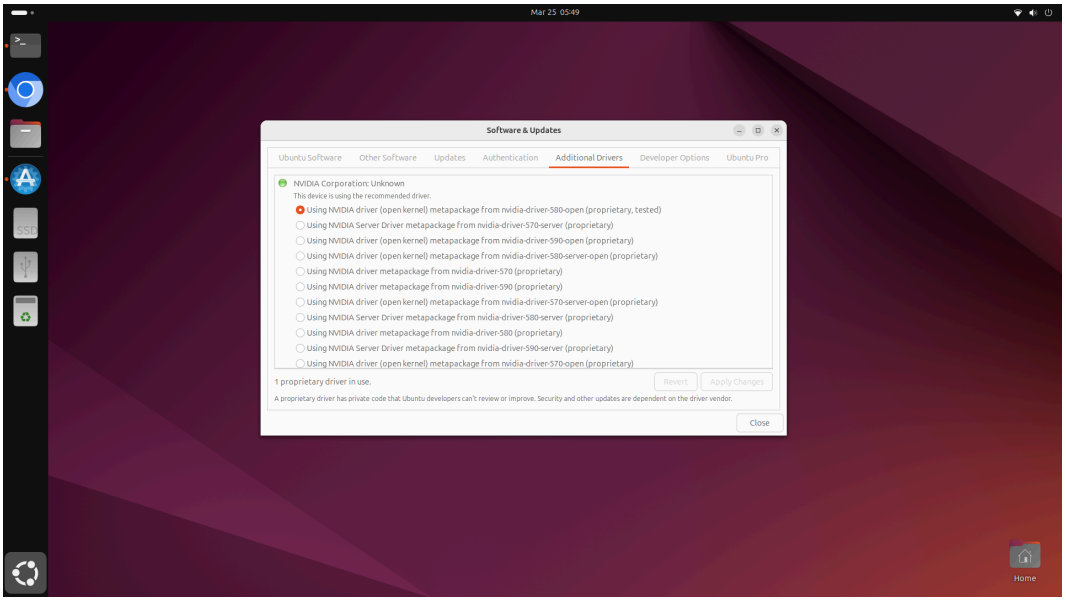


Figure. NVIDIA driver selected in Additional Drivers.

AMD

AMD graphics on Ubuntu often work well through open-source drivers such as *Radeon* or related open stacks, which the reference material describes as already adequate for many normal tasks and even decent acceleration. But it also explains that some users may still want the proprietary route when specific performance or vendor tools are needed.

For most students and office users, the open AMD path is often completely fine. That is an important lesson: **not every machine needs a proprietary graphics stack to be productive**. In Ubuntu Desktop, stable and functional often matters more than chasing theoretical performance.

CLI example: identify AMD graphics adapter

```
lspci | grep -i vga
```

Example output:

```
03:00.0 VGA compatible controller: Advanced Micro
Devices, Inc. [AMD/ATI] Radeon RX 560X
```

CLI example: inspect renderer information

```
glxinfo | grep -i render
```

Example output:

```
direct rendering: Yes
OpenGL renderer string: AMD Radeon RX 560X (RADV
POLARIS11)
```

The reference material specifically recommends `glxinfo | grep render` as a way to inspect graphics capability after installing `mesa-utils`.

Broadcom and Realtek Wi-Fi

Wireless networking is a major post-install concern on laptops, especially when Wi-Fi hardware is recognized only partially or not at all. The Ubuntu Academy outline explicitly names **Broadcom** and **Realtek** Wi-Fi as practical driver topics. In real desktop administration, this is extremely relevant because a system that cannot join a network cannot easily receive updates, language packs, or additional software.

A practical pattern for Wi-Fi troubleshooting is:

- identify the chipset
- confirm whether a driver is already loaded
- use **Additional Drivers** if Ubuntu offers one
- install the needed firmware or package once temporary internet is available

CLI example: identify wireless hardware

```
lspci -nnk | grep -A3 -i network
```

Example output:

```
02:00.0 Network controller: Broadcom Inc. and
subsidiaries BCM43142 802.11b/g/n
        Subsystem: Dell Device 0017
        Kernel modules: wl
```

Or on another machine:

```
03:00.0 Network controller: Realtek Semiconductor Co.,
Ltd. RTL8821CE 802.11ac PCIe Wireless Network Adapter
        Kernel driver in use: rtw_8821ce
        Kernel modules: rtw88_8821ce
```

CLI example: see whether a wireless interface exists

```
nmcli device status
```

Example output:

```
DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     CampusNet
lo          loopback  unmanaged     --
```

If no Wi-Fi device appears, the issue may be firmware, a missing module, or blocked hardware.

A useful real-world trick for learners is to keep a temporary **Ethernet cable** or **USB tethering** option available during setup. That gives the machine internet access so missing firmware or recommended drivers can be installed more easily.

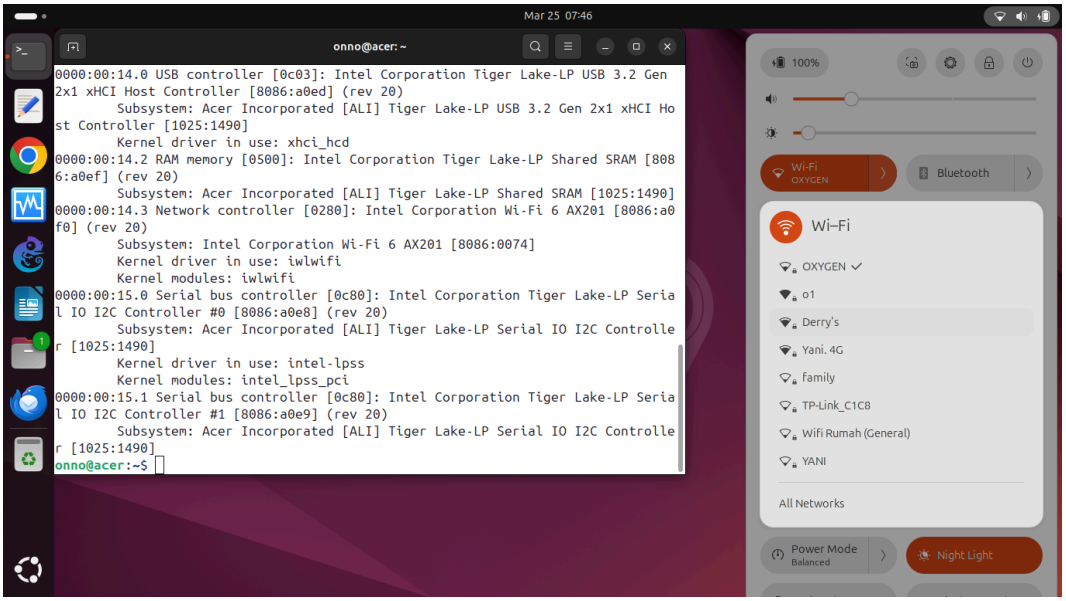


Figure. Showing lspci -nnk output and Wi-Fi connection

Language Packs, Timezone, and Input Methods

A good desktop should feel local, natural, and comfortable. Post-installation is the right time to confirm the display language, region, keyboard layout, timezone, and input methods. Some Ubuntu-based desktop experiences even highlight post-install configuration as including **language support** and user guidance, showing how important localization is for everyday usability.

This matters more than many beginners expect. A technically correct installation can still feel “wrong” if the clock is inaccurate, the keyboard layout is mismatched, or the system language does not support the reader’s normal workflow. Ubuntu Desktop is not only about operating a machine; it is about making the machine work naturally for the human using it.

A practical post-install checklist includes:

- confirm **display language**
- install missing **language packs**
- set the correct **timezone**
- verify **keyboard layout**
- add input methods if multiple languages are needed

CLI example: display current locale

```
locale
```

Example output:

```
LANG=en_US.UTF-8
LANGUAGE=en_US:
LC_TIME=en_US.UTF-8
LC_NUMERIC=en_US.UTF-8
LC_ALL=
```

CLI example: display time and timezone

```
timedatectl
```

Example output:

```
   Local time: Fri 2026-03-14 08:32:45 WIB
Universal time: Fri 2026-03-14 01:32:45 UTC
      RTC time: Fri 2026-03-14 01:32:44
     Time zone: Asia/Jakarta (WIB, +0700)
```

System clock synchronized: yes

CLI example: change timezone

```
sudo timedatectl set-timezone Asia/Jakarta
```

Typical result:

```
(no output if successful)
```

This is a good example of how desktop settings and system commands complement each other.

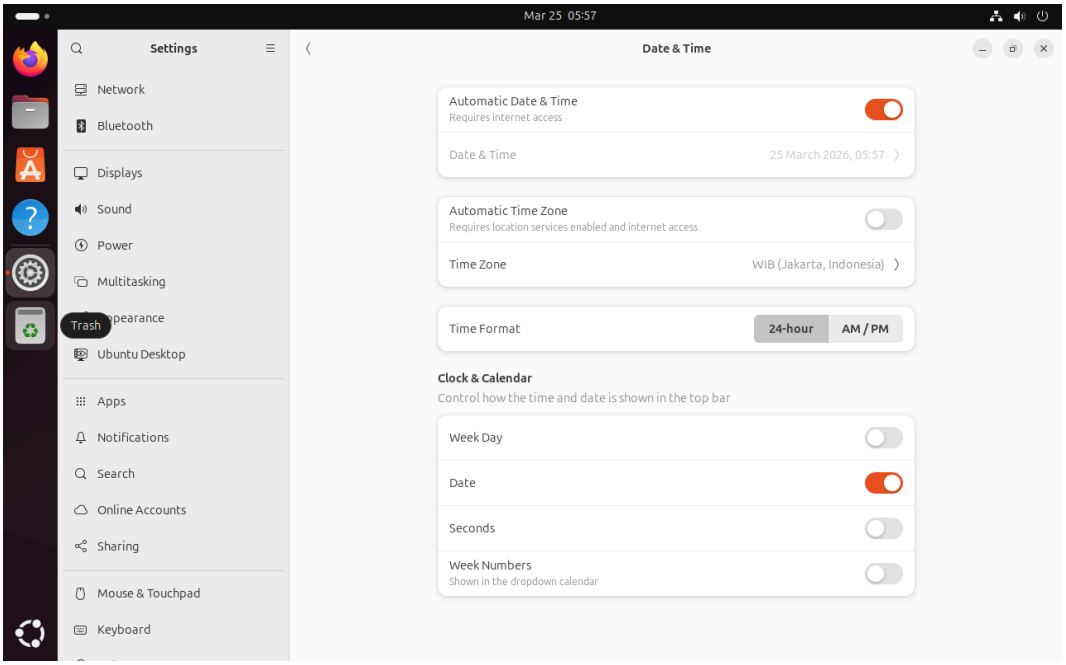


Figure. Setting Date, Time, Time Zone.

Enabling Extra Multimedia Support

A freshly installed Ubuntu system may already handle many common desktop tasks well, but multimedia support can be improved further when additional codecs or related packages are enabled. The Ubuntu Linux Bible specifically explains that Ubuntu makes it easy to select extra codecs during installation, especially for formats such as MP3, and that media applications may require those extras for broader format support. It also notes that tools like Rhythmbox can be extended with plug-ins and codecs for richer playback.

This is an important desktop lesson: not every media format is fully available in a default installation, especially where patents, licensing, or packaging choices affect distribution. Ubuntu gives users a practical path to add what they need.

A good user-focused explanation is this: the desktop is like a music room with speakers already installed, but a few instruments may still need to be brought in before the room handles every performance perfectly.

CLI example: install extra multimedia support

```
sudo apt install ubuntu-restricted-extras
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  ubuntu-restricted-extras gstreamer1.0-plugins-ugly
ttf-mscorefonts-installer
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 7,842 kB of archives.
```

This commonly adds useful codecs, fonts, and playback support for everyday desktop use.

CLI example: install a multimedia player

```
sudo apt install vlc
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  vlc vlc-bin vlc-plugin-base
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
```

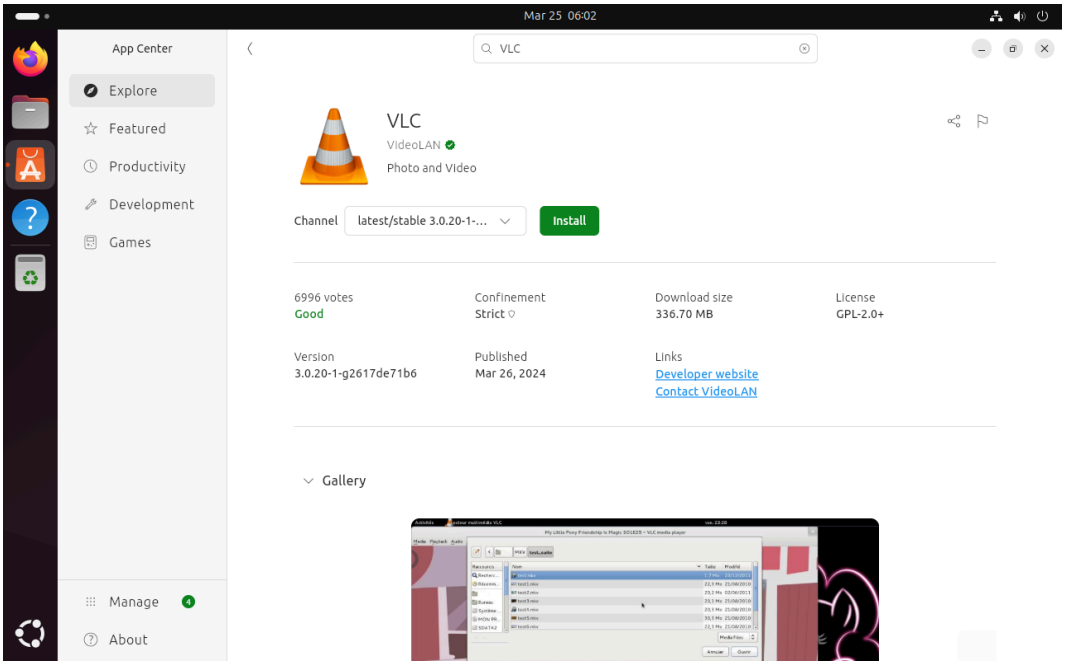


Figure. VLC Installation

Essential First Applications to Install

Ubuntu Desktop becomes much more personal after the first few applications are installed. The Ubuntu Linux Bible explains that Ubuntu is connected to its software repositories as soon as networking is available, allowing users to search and install thousands of packages through the graphical software tool or package commands. It also encourages users to install common desktop applications early so the desktop becomes effective for real daily work.

The reference material also gives examples of desktop applications and software availability across the Ubuntu ecosystem, including browsers, office tools, multimedia applications, email tools, cloud utilities, and common communication apps.

A practical starter set for many users includes:

- **web browser** if an alternative to the default is needed
- **office tools** for writing and spreadsheets
- **media tools** such as VLC
- **communication tools** such as messaging or meeting software
- **cloud sync tools** if part of the user workflow
- **developer tools** such as Git or VS Code if needed
- **ebook or PDF tools** depending on study habits

CLI example: install some useful starter applications

```
sudo apt install vlc git gnome-tweaks flatpak
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  flatpak git gnome-tweaks vlc
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
```

CLI example: install a Snap application

```
sudo snap install kolourpaint
```

Example output:

```
kolourpaint 25.04.1 from KDE installed
```

This command and result mirror the Snap workflow shown in the Ubuntu Linux Bible.

CLI example: enable Flatpak support

```
sudo apt install flatpak
flatpak remote-add --if-not-exists flathub
https://flathub.org/repo/flathub.flatpakrepo
```

Example output:

```
Flatpak system operation complete.
```

The Ubuntu Linux Bible also highlights Flatpak as an increasingly important sandboxed app option on Ubuntu.

Making Ubuntu Ready for Daily Desktop Use

This final stage is about turning a technically installed system into a comfortable working environment. The reference materials repeatedly show Ubuntu as a platform for browsing, office work, multimedia, productivity, and customization, backed by software repositories and graphical tools. They also frame Ubuntu as a system that can be made ready quickly after installation for real work, study, and development.

To make Ubuntu Desktop truly ready for daily use, readers should confirm the following:

- the internet connection is stable
- the system is updated
- the correct graphics and Wi-Fi drivers are in place
- language, keyboard, and timezone feel natural
- multimedia support is sufficient
- a small set of essential applications is installed
- the user understands where to get more software safely through repositories, Snaps, or Flatpaks

A good illustration is this: Ubuntu is like a newly opened workshop. The room is built, the lights are on, and the floor is clean. But only after the tools are placed on the wall, the network is connected, and the equipment is tested does it become a place where serious work can happen.

CLI example: quick readiness checklist

```
echo $XDG_CURRENT_DESKTOP
nmcli device status
timedatectl
apt list --upgradable
```

Example output:

```
ubuntu:GNOME

DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     OfficeNet
lo          loopback  unmanaged     --

                  Local time: Fri 2026-03-14 08:45:12 WIB
                   Time zone: Asia/Jakarta (WIB, +0700)

Listing... Done
```

A clean output here usually means the system is in good shape for normal desktop use.

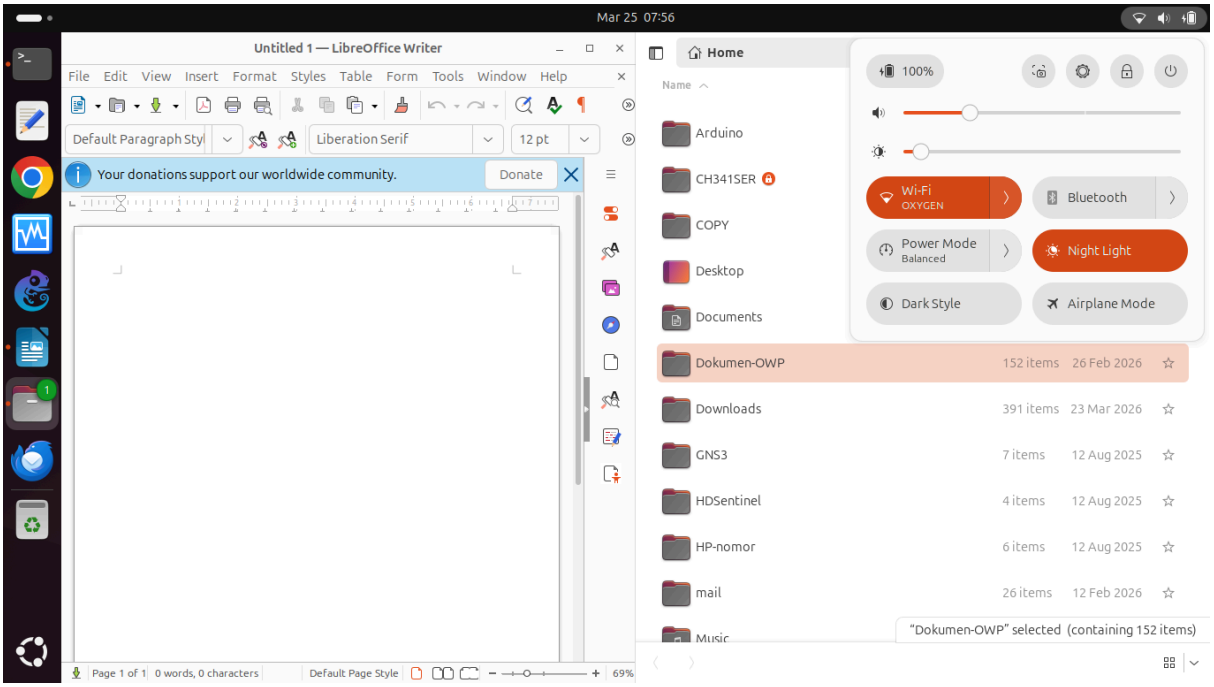


Figure. Completed Ubuntu desktop ready for daily use.

Guided Post-Install Activity

Readers can turn this chapter into a short practical lab:

- connect Ubuntu Desktop to **Wi-Fi or Ethernet**
- run `sudo apt update && sudo apt upgrade -y`
- open **Additional Drivers** and review hardware options
- verify graphics with `lspci | grep -i vga`
- verify wireless hardware with `lspci -nnk | grep -A3 -i network`
- confirm locale and timezone with `locale` and `timedatectl`
- install `ubuntu-restricted-extras`
- install at least three daily-use applications
- open the App Grid and confirm the tools appear
- reboot once and verify everything still works smoothly

This kind of guided practice turns post-install setup into a repeatable professional habit.

Chapter Reflection

Post-installation work is where Ubuntu Desktop becomes genuinely useful. A newly installed system must be connected to the internet, updated, and checked for the right hardware support. Some machines will work perfectly with open drivers, while others benefit from proprietary NVIDIA, AMD, or Broadcom/Realtek-related drivers. Localization settings such as language, keyboard layout, and timezone help the system feel natural for its user. Extra multimedia support expands everyday playback and media compatibility. Finally, a careful first set of applications transforms Ubuntu from a clean installation into a real desktop environment ready for study, productivity, and daily professional use.

Key Takeaways

- **Internet access is the gateway to updates, drivers, and software repositories.**
- **Updating immediately after installation is one of the best Ubuntu habits.**
- **Proprietary drivers should be installed when they solve real performance or compatibility needs.**
- **Broadcom and Realtek Wi-Fi support is an important practical topic for laptop users.**
- **Language packs, timezone, and keyboard settings make the system feel local and comfortable.**
- **Extra codecs and multimedia tools improve daily desktop usability.**
- **A small, well-chosen set of applications is enough to make Ubuntu productive from day one.**

Part III — Applications and Software Management

Chapter 6 — Managing Applications on Ubuntu Desktop

One of Ubuntu Desktop's greatest strengths is that software management is both **powerful** and **flexible**. A user can install applications graphically through the **App Center**, from the command line with **APT**, directly from local **.deb** files, or through newer package ecosystems such as **Snap** and **Flatpak**. The Ubuntu Academy outline makes this one of the heaviest focus areas in the book, highlighting **APT basics, .deb packages, Snap, Flatpak, App Center usage, safe third-party sources, and cleanup tasks** as core desktop administration skills.

A useful way to understand this chapter is to imagine Ubuntu as a well-organized city. The official repositories are the trusted supermarkets. A local `.deb` file is like bringing home a package directly from a shop. Snap and Flatpak are more like self-contained delivery boxes that include most of what they need inside. PPAs and third-party repositories are specialty stores: sometimes very useful, but they must be chosen carefully. Once readers understand these sources, Ubuntu software management stops feeling confusing and starts feeling logical.

Understanding Ubuntu Software Sources

Ubuntu applications can come from several different places, and each source has its own strengths. The most traditional and most trusted source is the **Ubuntu repository system**, which works through **APT** and the Debian package format. These repositories are integrated into Ubuntu and are the normal source for thousands of applications and updates. The reference materials emphasize that once Ubuntu is connected to the internet, it is ready to communicate with these repositories so users can install and update software safely and efficiently.

Ubuntu Desktop also supports **Snap**, which Canonical integrates closely into the Ubuntu ecosystem, and **Flatpak**, which is increasingly used for sandboxed desktop applications across Linux. In addition, users may install software directly from downloaded `.deb` files or, in some cases, from PPAs and third-party repositories. This is why Ubuntu is so attractive for desktop use: there is rarely only one path to getting software installed. The real skill is knowing **which method is most appropriate for which situation**.

A practical decision model looks like this:

- use **APT/repositories** for trusted, integrated system software
- use **App Center** for easy graphical installation
- use **Snap** for convenient, self-contained apps tightly integrated with Ubuntu
- use **Flatpak** when the application is distributed well through Flathub or other Flatpak sources
- use **local `.deb` files** only when needed and only from trusted sources
- use **PPAs/third-party repositories** carefully and sparingly

GUI App Installation with App Center / Ubuntu Software

For many desktop users, the easiest starting point is the **App Center** or Ubuntu Software interface. The Ubuntu Academy outline explicitly includes GUI installation and updates here, and the reference materials describe using the software center to search for applications, review details, and install them with a few clicks. This is especially helpful for beginners because it reduces command-line pressure while still teaching software selection habits.

The graphical approach is excellent for:

- browsing available applications
- reading descriptions before installing
- checking screenshots and user-facing details
- installing desktop apps without memorizing package names
- discovering whether an app is offered as a repository package or a Snap

The reference material gives several examples of this workflow, such as installing applications like VLC or Telegram through Ubuntu Software. This reinforces an important lesson: **desktop package management is not only a terminal skill**. On Ubuntu Desktop, GUI and CLI methods complement each other. A student might browse in App Center first, then later learn the exact terminal command for the same package.

A good illustration is to compare App Center with a bookstore. You can browse covers, descriptions, and editions before choosing what to take home. It is slower than typing an exact command, but it is often friendlier for discovery.

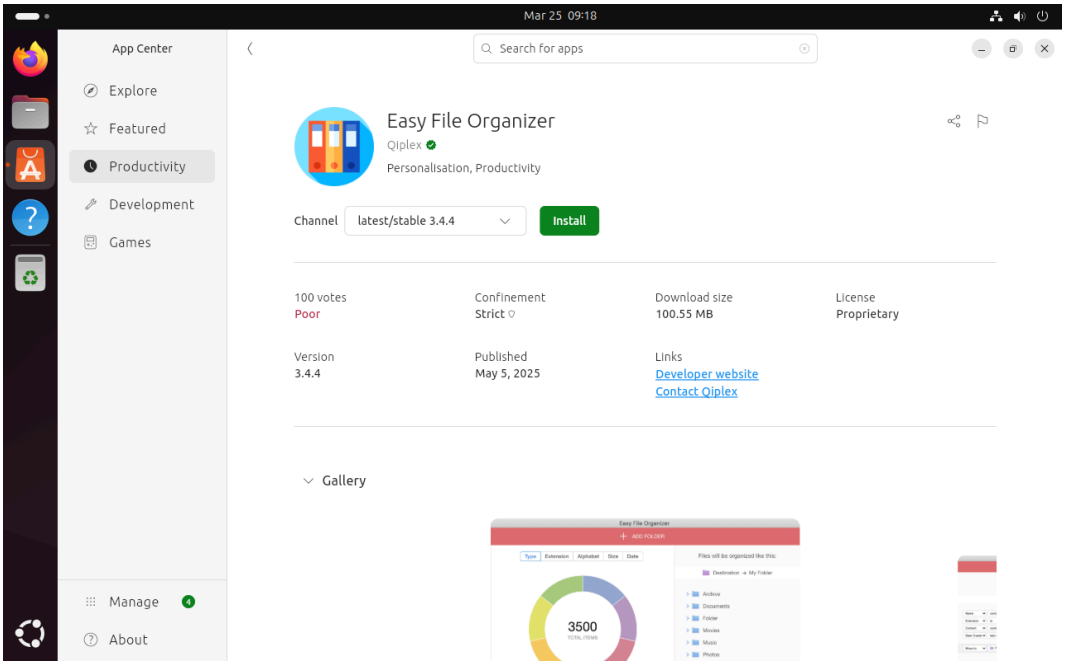


Figure. Detail Apps Page in App Center

APT Fundamentals

APT is the traditional and foundational package management interface for Ubuntu and other Debian-based systems. The Ubuntu Academy outline places APT at the center of package management learning, including `apt update`, `apt upgrade`, `apt install`, `apt remove`, and `apt search`. These commands are essential because they teach readers how Ubuntu's repository-based software model works.

A useful metaphor is that APT is like the logistics department of Ubuntu. It checks the latest package lists, compares versions, downloads needed files, resolves dependencies, and installs software in an orderly way. Once readers learn a few core APT commands, they gain a level of control and visibility that graphical tools alone do not always provide.

apt update

This command refreshes local package metadata so Ubuntu knows what versions are available from configured repositories. It does **not** install updates by itself. It simply updates the package index.

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble-updates
InRelease
Hit:3 http://security.ubuntu.com/ubuntu noble-security
InRelease
Reading package lists... Done
```

This is often the first command to run before installing or upgrading software.

apt upgrade

After refreshing metadata, `apt upgrade` installs the newer versions of already installed packages when available.

```
sudo apt upgrade -y
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Calculating upgrade... Done
The following packages will be upgraded:
  firefox linux-firmware
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

This is part of healthy Ubuntu maintenance and keeps the desktop current.

apt install

This installs a package from the configured repositories.

```
sudo apt install vlc
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  vlc
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
```

The references repeatedly show this style of installation for Ubuntu applications, including tools like VLC and Synaptic.

apt remove

This removes an installed package while generally leaving user configuration files behind.

```
sudo apt remove vlc
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED:
  vlc
0 upgraded, 0 newly installed, 1 to remove and 0 not
upgraded.
```

This is useful when uninstalling applications that are no longer needed.

apt search

This lets users search package names and descriptions from the repositories.

```
apt search telegram
```

Example output:

```
Sorting... Done
Full Text Search... Done
telegram-desktop/noble 4.15.2-1 amd64
  fast and secure messaging application
```

This is one of the fastest ways to discover package names before installing them.

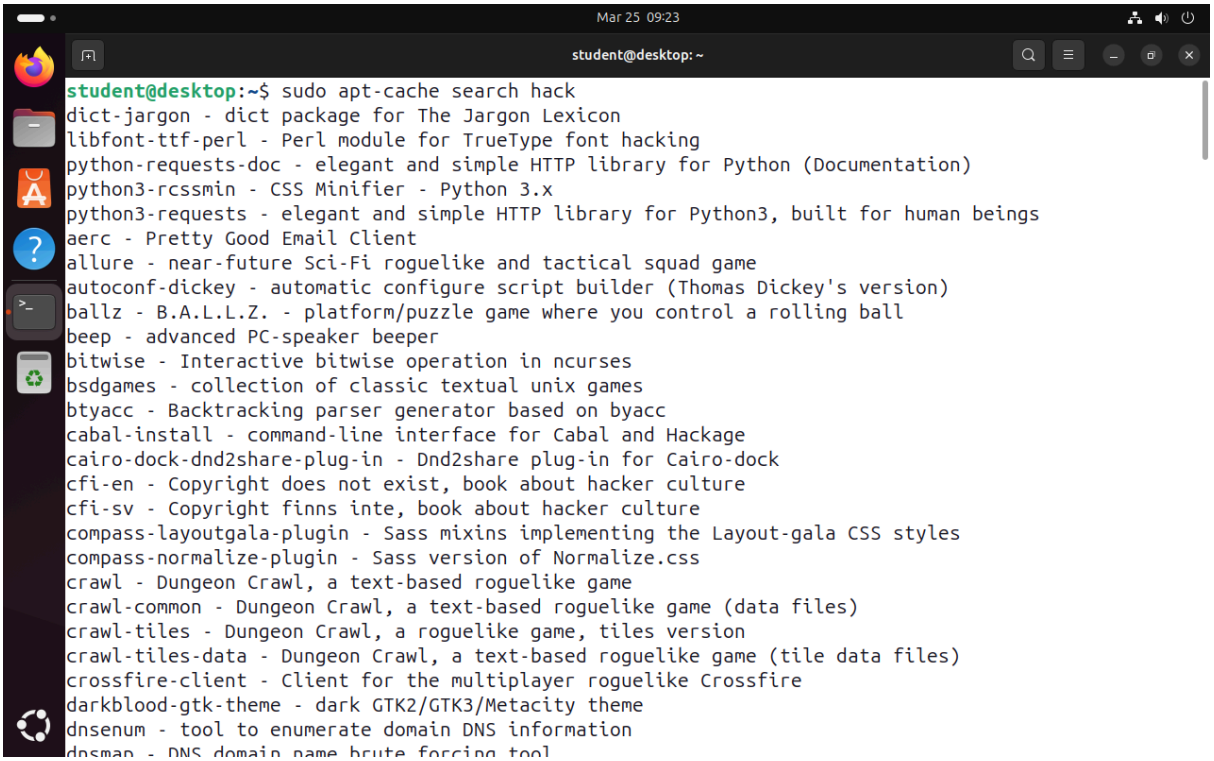


Figure. Result of apt-cache search

Working with .deb Packages

Although APT is usually the preferred way to install software, Ubuntu users sometimes download an application directly as a **`.deb` package**. This happens when software is distributed from a vendor website or when a newer or special build is not available in the standard repositories. The Ubuntu Linux Bible explains that **dpkg** is the traditional tool for handling local Debian packages already present on the machine.

A local `.deb` package is like receiving a boxed product directly instead of ordering through the supermarket. It can be useful, but the user must make sure the source is trustworthy and that dependencies are handled correctly. Some applications in the reference materials, such as Franz or Chrome-style installers, are discussed in this direct-download pattern.

dpkg

The Ubuntu Linux Bible explicitly shows using `dpkg -i` to install a local package and `dpkg -r` to remove it.

CLI example: install a local `.deb` package

```
sudo dpkg -i example-app_1.0_amd64.deb
```

Example output:

```
(Reading database ... 205231 files and directories
currently installed.)
Preparing to unpack example-app_1.0_amd64.deb ...
Unpacking example-app (1.0) ...
Setting up example-app (1.0) ...
```

CLI example: remove a package installed with dpkg

```
sudo dpkg -r example-app
```

Example output:

```
(Reading database ... 205240 files and directories
currently installed.)
Removing example-app (1.0) ...
```

CLI example: list installed packages with dpkg

```
dpkg -l | head
```

Example output:

```
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Name           Version        Architecture Description
ii  accountsservice 0.6.55-0ubuntu amd64       query and manipulate user account
information
ii  acl             2.3.1-1        amd64       access control list utilities
```

The references also discuss `dpkg -l` as a way to inspect installed packages.

GUI helpers

The Ubuntu Academy outline mentions **gdebi for GUI**. GUI helpers can make local `.deb` installation easier because they present package details visually and often help with dependency handling. The beginner reference also points to GUI-oriented package helpers such as **Synaptic**, reinforcing that Debian packages can be managed through both graphical and terminal methods.

Snap Packages

Snap is a self-contained packaging system strongly integrated into Ubuntu. The reference materials explain that Snaps bundle their own dependencies, can be managed by GUI or CLI tools, and are designed to work in a more isolated environment than traditional packages. The Ubuntu Academy outline specifically highlights `snap install`, `snap search`, `snap remove`, `snap refresh`, `snap list`, and the idea of **classic vs strict confinement**.

The main idea is simple: a Snap carries much of what it needs with it. This can reduce dependency conflicts and make distribution easier across systems, though the trade-offs may include larger disk use or slower startup for some applications. The beginner reference specifically notes this larger disk footprint and different behavior compared with normal `.deb` packages.

Search

```
snap find telegram
```

Example output:

```
Name              Version   Publisher   Notes    Summary
telegram-desktop  5.0.1     canonical✓  -        Official Telegram desktop client
```

The beginner reference even shows a pattern using `snap find | grep Telegram`.

Install

```
sudo snap install telegram-desktop
```

Example output:

```
telegram-desktop 5.0.1 from Canonical✓ installed
```

Refresh

```
sudo snap refresh
```

Example output:

```
All snaps up to date.
```

Remove

```
sudo snap remove telegram-desktop
```

Example output:

```
telegram-desktop removed
```

List

```
snap list
```

Example output:

```
Name              Version   Rev   Tracking       Publisher   Notes
core22            20250214  1880  latest/stable  canonical✓  base
snapd             2.67      23545 latest/stable  canonical✓  snapd
telegram-desktop  5.0.1     7123  latest/stable  canonical✓  -
```

Confinement concepts

The Ubuntu Academy outline mentions **classic vs strict confinement**, and the reference materials describe Snap and Flatpak as working in sandboxed or contained environments that reduce conflicts and can improve host protection. A simple way to explain this is:

- **strict confinement** keeps the app in a tighter sandbox with limited default access
- **classic confinement** gives the app broader access when that is necessary for its role

An easy illustration is to think of strict confinement as visiting a lab through controlled doors, while classic confinement is more like a staff member with wider building access.

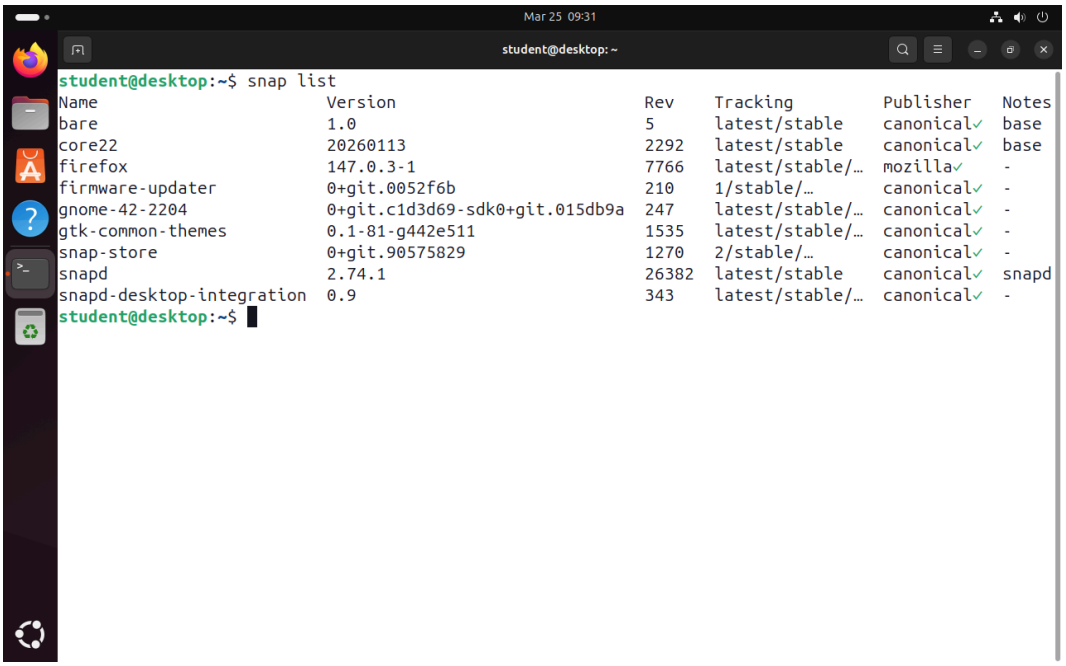


Figure. Output of snap list

Flatpak and Flathub

Flatpak is another self-contained application format widely used for Linux desktop applications. The reference materials explain that Flatpak can provide sandboxed installations, reduce dependency conflicts, and allow software to run across multiple Linux distributions. The Ubuntu Academy outline specifically includes **installing Flatpak, adding Flathub, and using flatpak install/run/update**.

If Snap is strongly tied into Ubuntu's ecosystem, Flatpak is often associated with broader cross-distribution desktop packaging, especially through **Flathub**. Many Linux users appreciate Flatpak for desktop applications that are updated independently from the distribution's core packages. The beginner reference explicitly discusses enabling Flatpak support and installing apps through Flatpak or GNOME software integration.

Installation

First install Flatpak support:

```
sudo apt install flatpak
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  flatpak
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
```

Then add Flathub:

```
flatpak remote-add --if-not-exists flathub
https://flathub.org/repo/flathub.flatpakrepo
```

Example output:

```
Flatpak system operation complete.
```

Integration

Some Ubuntu desktop workflows also integrate Flatpak with graphical software tools, making Flatpak applications visible in GUI software centers. The beginner reference mentions GNOME software center support for Flatpak.

Updating apps

To install an app:

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

Example output:

```
Looking for matches…
Required runtime for org.gimp.GIMP/x86_64/stable found in remote flathub
Proceed with these changes to the system installation? [Y/n]: y
```

To update Flatpak applications:

```
flatpak update
```

Example output:

```
Looking for updates…
Nothing to do.
```

This gives users a modern desktop-app workflow that is separate from APT but still very practical.

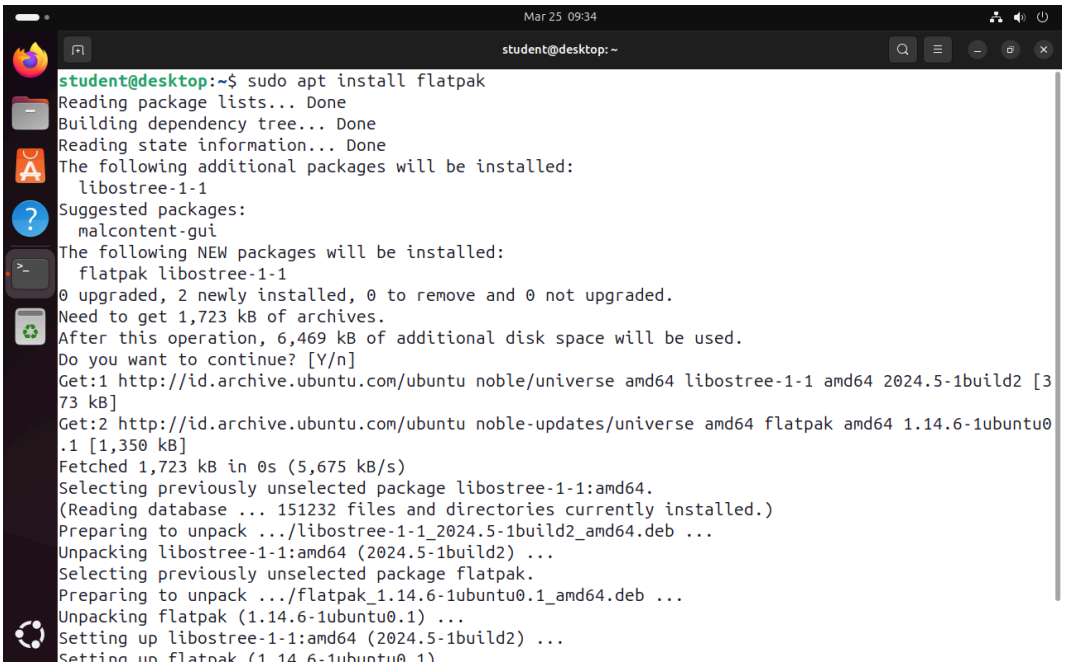


Figure. Install flatpak

PPAs and Third-Party Repositories Safely

Sometimes software is not available in the official repositories or the packaged version is older than the user wants. In those cases, users may consider a **PPA** or another third-party repository. The Ubuntu Academy outline includes this topic, but it also clearly frames it in terms of **safe handling**. That caution is strongly supported by the Ubuntu Linux Bible, which explicitly warns users to trust only sources they know, because these are **not curated the same way as the mainstream repositories**.

That warning is important. A PPA can be useful, but it also changes the trust relationship of the system. A repository with poor maintenance, broken keys, or outdated packages can create security and stability risks. The Ubuntu Linux Bible even gives an example where a repository key problem makes the repository unsuitable to trust.

CLI example: add a PPA

```
sudo apt-add-repository ppa:example/project
sudo apt update
```

Example output:

```
Press [ENTER] to continue or Ctrl-c to cancel adding it.
Get:1 http://ppa.launchpad.net/example/project/ubuntu
noble InRelease
Reading package lists... Done
```

CLI example: remove a broken or unwanted PPA source file

```
ls /etc/apt/sources.list.d/
sudo rm
/etc/apt/sources.list.d/example-project-noble.list
sudo apt update
```

This follows the practical pattern described in the Ubuntu Linux Bible for removing an unwanted repository source.

A good safety rule for readers:

- prefer **official Ubuntu repositories** first
- use **Snaps or Flatpaks** from trusted channels when appropriate
- add a **PPA only when there is a clear reason**
- remove old, broken, or untrusted repositories promptly

Cleaning Unused Packages and Managing Disk Space

Good package management is not only about installing software. It is also about **removing clutter** and keeping the system tidy. The Ubuntu Academy outline explicitly includes **apt autoremove** and **snap auto-cleanup** as practical maintenance tasks. This is especially important on laptops, small SSDs, virtual machines, and teaching lab systems where storage space matters.

A clean system is usually easier to maintain, easier to back up, and less confusing for new users. Old dependencies, unused applications, and unnecessary cached data can quietly consume space over time. This is why package cleanup belongs in a desktop book, not only in a server book.

CLI example: remove unneeded dependency packages

```
sudo apt autoremove
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED:
  libexample1 old-runtime-helper
After this operation, 148 MB disk space will be freed.
```

CLI example: check disk usage

```
df -h
```

Example output:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p2  236G   28G  197G  13% /
tmpfs           1.6G  2.1M  1.6G   1% /run
```

CLI example: view installed Snaps

```
snap list
```

And where appropriate, Snap cleanup settings can also be tuned as noted in the Ubuntu Academy outline.

A nice illustration is to compare this to a study desk. Installing apps is like putting tools on the desk. Cleanup is deciding which tools are still useful and clearing away the ones that only waste space.

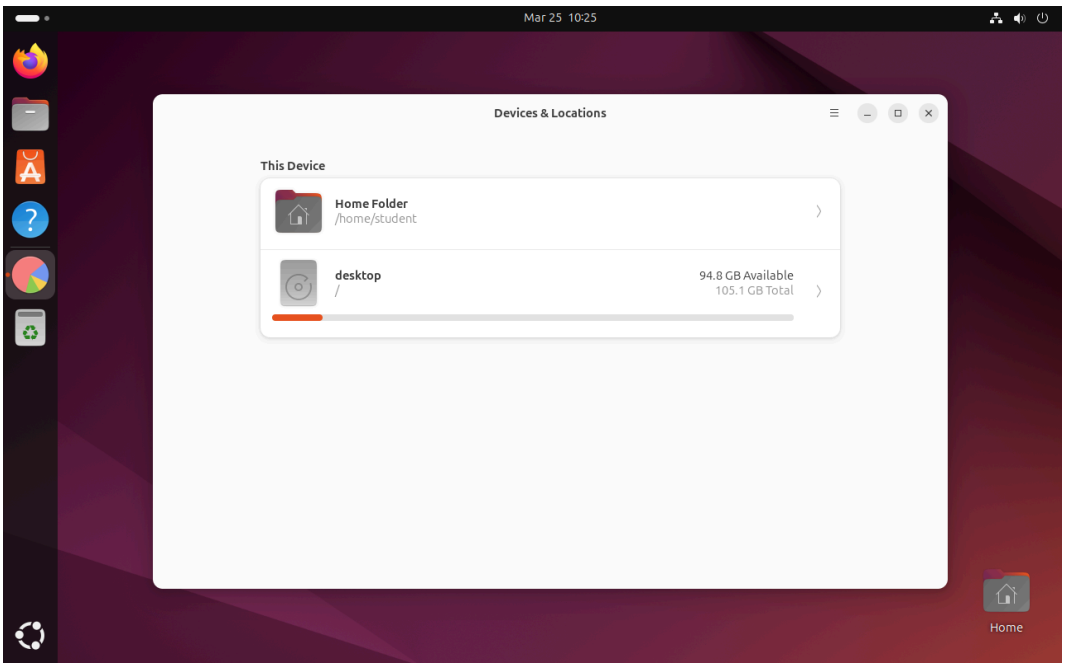


Figure. Disk Usage Analysis

Guided Application Management Lab

Readers can turn this chapter into a practical exercise:

- search for an app in **App Center**
- install one app with **APT**
- install one app as a **Snap**
- enable **Flatpak** and install one app from **Flathub**
- inspect installed packages with `dpkg -l | head`
- install or inspect a local `.deb` package
- run `apt search` for a package they are curious about
- remove one test package
- run `sudo apt autoremove`
- review software sources and identify whether any unnecessary third-party repository is enabled

This kind of lab builds real confidence because it teaches not only how to install software, but how to think about Ubuntu's software ecosystem as a whole.

Chapter Reflection

Managing applications on Ubuntu Desktop is one of the most practical and empowering skills a desktop user can learn. Ubuntu gives users several pathways: official repositories through **APT**, visual discovery through **App Center**, direct local installation with `.deb` files and **dpkg**, and modern self-contained formats through **Snap** and **Flatpak**. Each method serves a real purpose. At the same time, package management also requires judgment: readers must choose trusted sources, treat PPAs carefully, and clean up unused software regularly. Once these ideas become familiar, Ubuntu Desktop becomes a flexible and reliable environment for both everyday productivity and deeper technical learning.

Key Takeaways

- **Ubuntu supports multiple software sources**, and each has a valid role.
- **App Center** is excellent for visual browsing and beginner-friendly installation.
- **APT** remains the core package management tool for Ubuntu repositories.
- `dpkg` is used for local `.deb` packages already on the machine.
- **Snap** and **Flatpak** provide sandboxed, self-contained application models.
- **PPAs and third-party repositories require caution and trust evaluation.**
- **Cleanup commands such as `apt autoremove` help keep Ubuntu lean and healthy.**

Chapter 7 — Everyday Ubuntu Desktop Applications

A desktop operating system becomes meaningful when it supports **real daily work**. After installation, updates, and application management basics, the next step is learning how Ubuntu Desktop serves as a platform for browsing, communication, writing, media, creativity, reading, and collaboration. The attached references consistently show Ubuntu as a practical everyday desktop, not only a technical learning system. They highlight browsers, office tools, media players, email applications, image editors, social tools, and multiple package formats for installing them.

A helpful way to think about this chapter is to imagine Ubuntu Desktop as a well-equipped study and work room. One corner is for the web, another for writing and documents, another for music and video, another for notes and reading, and another for collaboration. The operating system itself is the room, but the applications are the tools placed on the desk. Choosing the right tools—and knowing how to install them wisely—is what turns Ubuntu into a serious daily workstation.

Web Browsers and Safe Browsing

A modern desktop without a browser is almost unimaginable. The Ubuntu references show that Linux users have several browser choices, with **Firefox** remaining one of the best-known and most established options on Ubuntu. The materials also mention **Chromium** as an important alternative, while **Chrome** appears in practical examples where users need broader media or service compatibility.

Firefox is a strong default because it is mature, capable, and well integrated into Linux desktops. The Complete Guide notes useful features such as syncing tabs and bookmarks across devices, which supports remote work and cross-device continuity. That matters for students and professionals who move between machines or want a consistent browsing environment.

Safe browsing is not only about which browser you choose. It is also about **habits**. Ubuntu Academy's desktop and security direction clearly fits a browsing mindset that includes:

- using trusted download sources
- avoiding suspicious browser prompts
- verifying websites before installing extensions or applications
- keeping the browser updated
- using sandboxed apps or trusted repositories whenever possible

A good illustration is to think of a browser as the main front door of your digital house. A strong door helps, but safe behavior still matters. You still need to check who is outside before opening it.

CLI example: install Firefox from Ubuntu repositories

```
sudo apt update
sudo apt install firefox
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
firefox is already the newest version
(1:1snap1-0ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 0 not
upgraded.
```

CLI example: search for Chromium

```
apt search chromium-browser
```

Example output:

```
Sorting... Done
Full Text Search... Done
chromium-browser/noble 1:124.0 amd64
  Chromium web browser
```

CLI example: launch the browser from Terminal

```
firefox &
```

Example output:

```
[1] 4821
```

This shows the browser launched as a background graphical process.

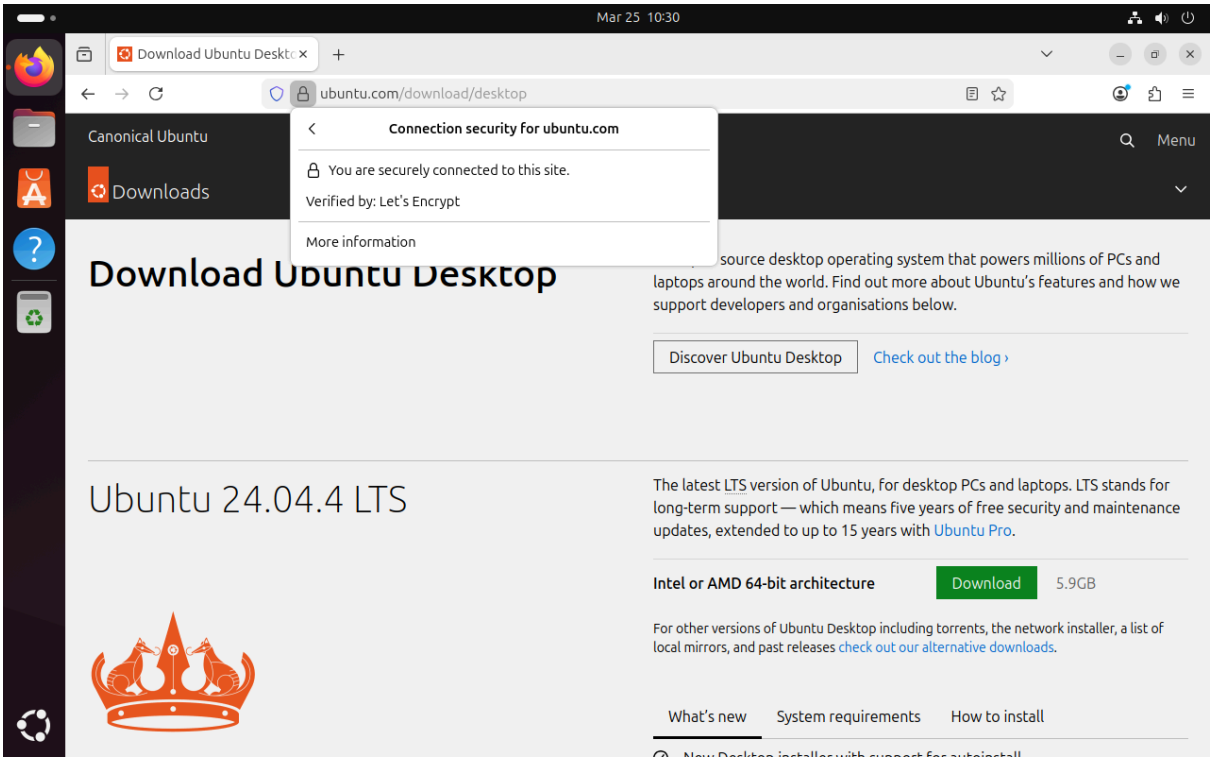


Figure. Secured Browsing HTTPS using Firefox.

Email and Communication Apps

Email remains one of the most important desktop activities in education and professional work. The reference materials show Ubuntu as a practical platform for everyday communication, including email setup and internet-based workflows. Ubuntu The Complete Guide explicitly includes a section on **setting up email**, while the browser and software references also show that Ubuntu supports a wide range of communication applications.

For many users, a desktop email application is useful because it creates a more focused work environment than relying only on browser tabs. It can also integrate better with notifications, attachments, and multi-account workflows. A communication application on Ubuntu is more than a message reader; it is often part of the daily productivity system.

A practical desktop communication toolkit may include:

- **email client** for academic or professional correspondence
- browser-based communication tools when needed
- chat or messaging platforms
- video meeting applications
- calendar-linked communication tools depending on workflow

A useful illustration is to compare email and communication apps to the reception desk of an office. Messages arrive there first, and from there the rest of the day's work is often organized.

CLI example: search for a desktop email client

```
apt search thunderbird
```

Example output:

```
Sorting... Done
Full Text Search... Done
thunderbird/noble 1:115.10.1 amd64
  Mozilla Thunderbird mail and news reader
```

CLI example: install Thunderbird

```
sudo apt install thunderbird
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  thunderbird
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
```

CLI example: launch the email client

```
thunderbird &
```

Example output:

```
[1] 5198
```

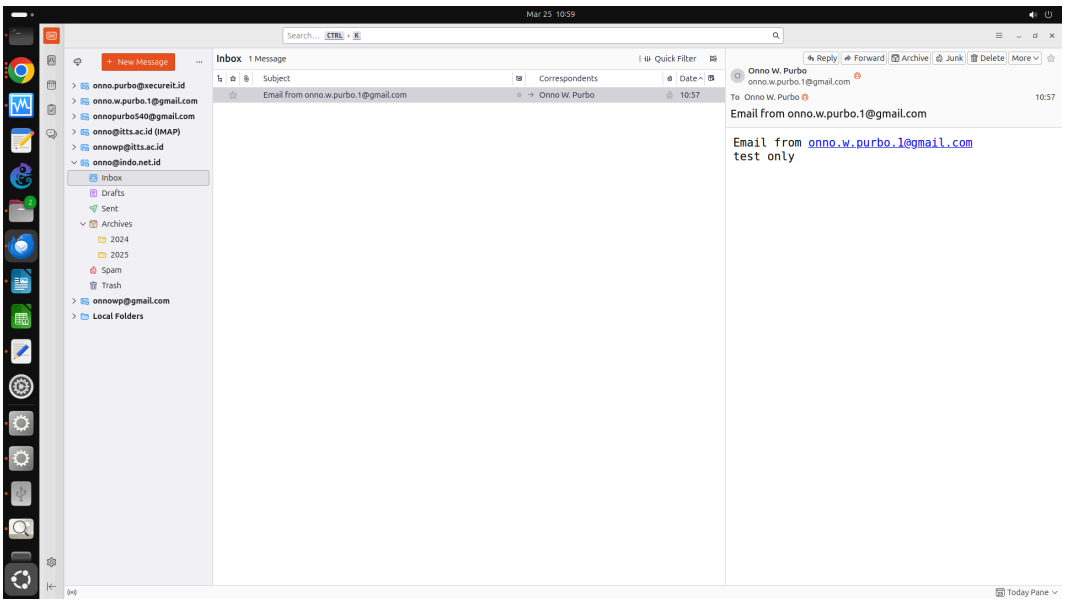


Figure. Thunderbird eMail client.

Office and Document Productivity

Ubuntu Desktop is fully capable of supporting writing, spreadsheets, presentations, and general office work. The reference materials point to Ubuntu as a practical replacement for common desktop workflows and emphasize that Linux desktop users can find equivalent applications for most major tasks typically associated with Windows or macOS systems.

Office productivity on Ubuntu matters because this is where many users truly test whether the desktop is ready for real life. A student needs reports. A lecturer needs presentation slides. A staff member needs forms, tables, and letters. A desktop that cannot support documents well will never feel complete.

Ubuntu commonly works well with office applications for:

- word processing
- spreadsheets
- presentations
- note preparation
- PDF export and document sharing
- cross-platform file compatibility

A simple way to explain this is to imagine the office suite as the main writing table of the desktop. If the browser is the front door, the office suite is where much of the actual thinking and output happens.

CLI example: search for LibreOffice

```
apt search libreoffice
```

Example output:

```
Sorting... Done
Full Text Search... Done
libreoffice/noble 4:24.2.2 amd64
  office productivity suite
```

CLI example: install LibreOffice

```
sudo apt install libreoffice
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  libreoffice libreoffice-writer libreoffice-calc
libreoffice-impress
0 upgraded, 4 newly installed, 0 to remove and 0 not
upgraded.
```

CLI example: start Writer

```
libreoffice --writer &
```

Example output:

```
[1] 5442
```

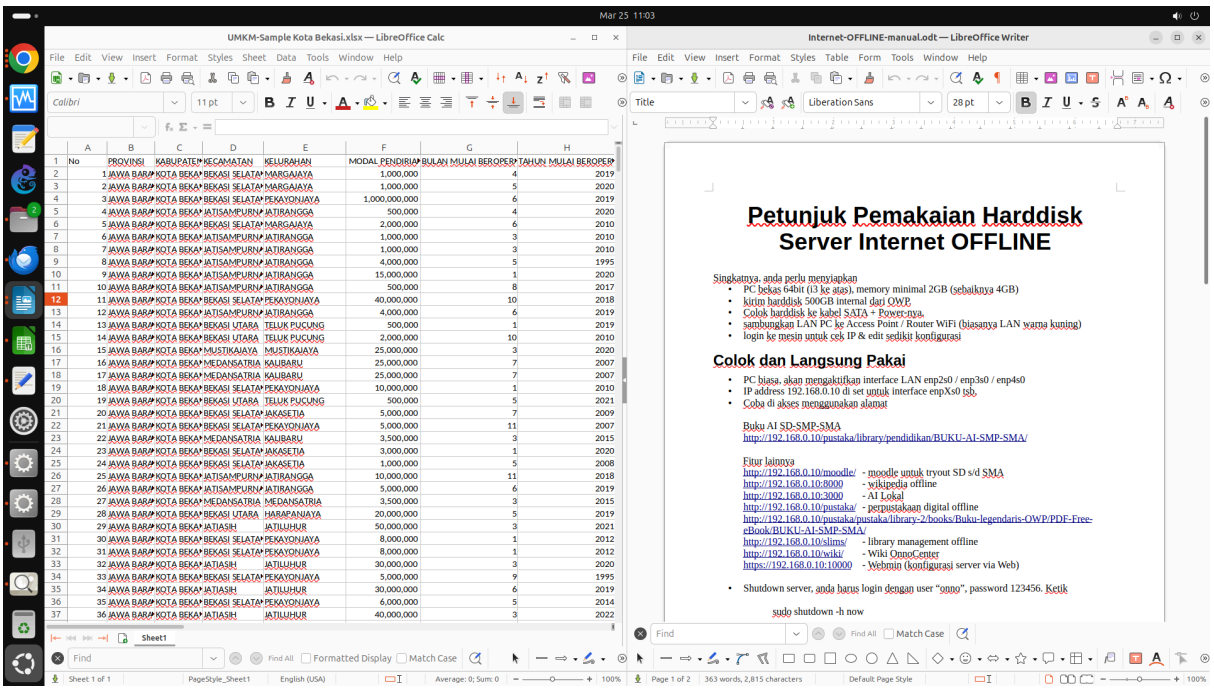


Figure. Libreoffice Word and Calc.

Media Playback: Music, Video, and Podcasts

A desktop should not only support work. It should also support enjoyment, focus, and creative energy. The Ubuntu references clearly present Ubuntu as a capable media desktop. Ubuntu The Complete Guide includes sections on films, music, and photos, while Ubuntu Linux Bible discusses **Rhythmbox** for music, podcasts, internet radio, and audio CDs. It also notes that plug-ins or codecs may be needed for some formats such as MP3.

Rhythmbox is presented as a music hub for Ubuntu Desktop. The materials describe it handling local audio libraries, podcasts, internet radio, and even plugin-based expansion. This makes it a good teaching example because it shows how Ubuntu applications often begin with a solid base and become richer through extensions.

For video playback, the references strongly recommend **VLC** as a must-have desktop media tool. Mastering Ubuntu explicitly presents VLC as easy to use and installable either via Snap or APT, while Ubuntu The Complete Guide also highlights it as probably the best Ubuntu video player.

A useful illustration is this: Rhythmbox is like the music shelf and radio system of your room, while VLC is the universal screen that plays almost anything you place in front of it.

CLI example: install VLC with APT

```
sudo apt update
sudo apt install vlc
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  vlc
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
```

This matches the repository-based VLC workflow described in the reference.

CLI example: install VLC as a Snap

```
sudo snap install vlc
```

Example output:

```
vlc 3.0.21 from VideoLAN✓ installed
```

This mirrors the Snap-based VLC method shown in the reference materials.

CLI example: launch Rhythmbox

```
rhythmbox &
```

Example output:

```
[1] 5610
```

CLI example: launch VLC

```
vlc &
```

Example output:

```
[1] 5664
```

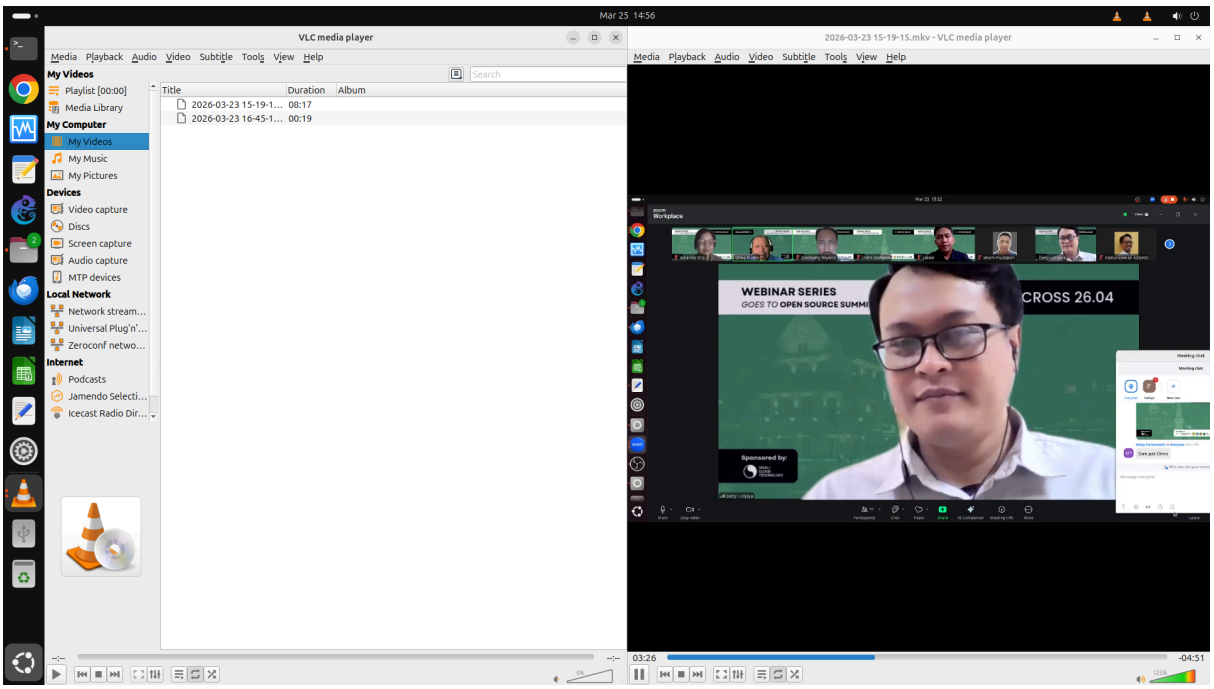


Figure. VLC playing Video.

Image Editing and Creative Tools

Ubuntu Desktop is not limited to office work and browsing. It is also a capable environment for visual creativity. The reference materials specifically mention **GIMP** as a major image-editing application and **Inkscape** as a graphics editor. They also mention creative tools such as **Kdenlive** and **Krita** in broader desktop-environment discussions, showing that Linux desktop users can build serious creative workflows.

This is important because many users assume Linux desktops are only for coding or system administration. The references challenge that assumption. Ubuntu can support image editing, vector work, and media creation through mature open-source tools.

For creative use, Ubuntu may include or support:

- **GIMP** for raster image editing
- **Inkscape** for vector graphics
- **Kdenlive** for video editing
- **Krita** for digital painting
- screenshot and screen-recording tools for tutorials or online teaching

A simple illustration is to think of Ubuntu's creative apps as a workshop table for art and media. The desk may be the same one used for writing in the morning, but by afternoon it can become a space for posters, diagrams, editing, and design.

CLI example: install GIMP

```
sudo apt install gimp
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  gimp
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

CLI example: install Inkscape

```
sudo apt install inkscape
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  inkscape
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
```

CLI example: launch GIMP

```
gimp &
```

Example output:

```
[1] 5902
```

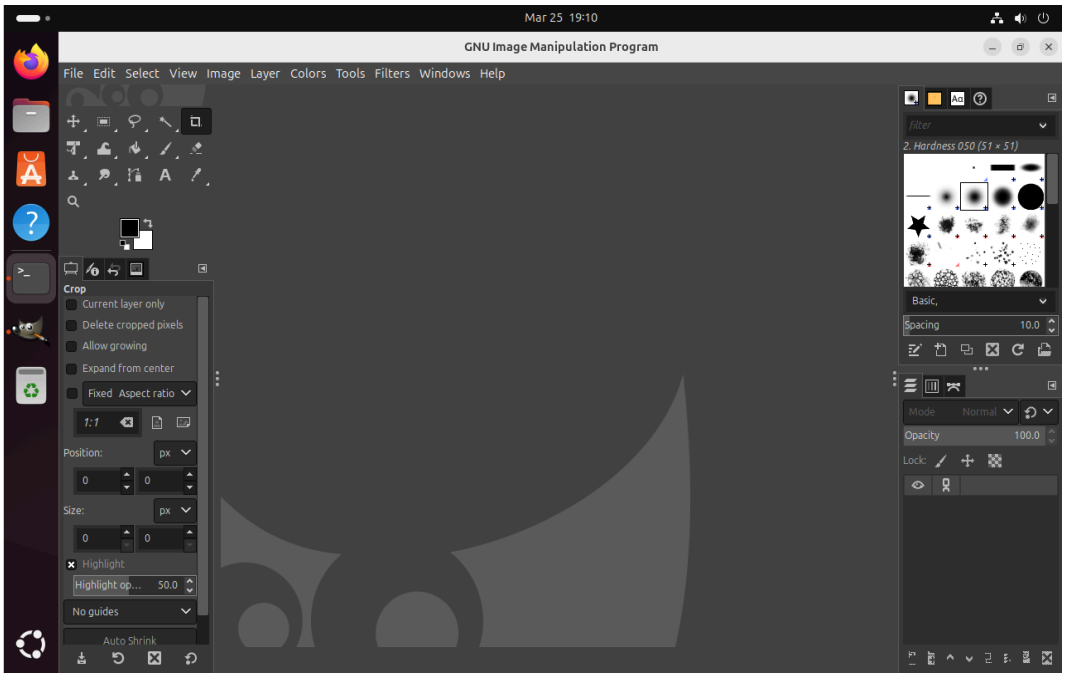


Figure. GIMP workspace

PDF, eBook, and Note-Taking Apps

A real desktop must support reading and annotation as well as writing. Students, researchers, and professionals spend much of their day opening PDFs, reading reference materials, checking manuals, or collecting notes. Even when the references do not frame this as a separate formal category, they clearly position Ubuntu as a document-friendly desktop system with strong file management and desktop-application equivalence to other operating systems.

PDF and note workflows matter because they sit at the intersection of reading and production. A user may browse a manual in PDF, extract information into notes, and then build a report or presentation from those notes. Ubuntu's application ecosystem supports this pattern well.

A practical daily reading toolkit may include:

- a **PDF viewer** for manuals and academic articles
- an **eBook reader** for longer-form digital books
- a **note-taking tool** for study, planning, and project drafts
- screenshot tools for capturing and referencing visual material

A useful illustration is to imagine these tools as the reading lamp and notebook on the corner of your desk. They may not be as glamorous as a browser or media player, but they are essential for serious study.

CLI example: search for a note-taking tool

```
apt search xournal
```

Example output:

```
Sorting... Done
Full Text Search... Done
xournalpp/noble 1.2.3 amd64
  handwriting notetaking software with PDF annotation support
```

CLI example: install a PDF-annotation or note-taking application

```
sudo apt install xournalpp
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  xournalpp
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

CLI example: launch a note-taking app

```
xournalpp &
```

Example output:

```
[1] 6044
```

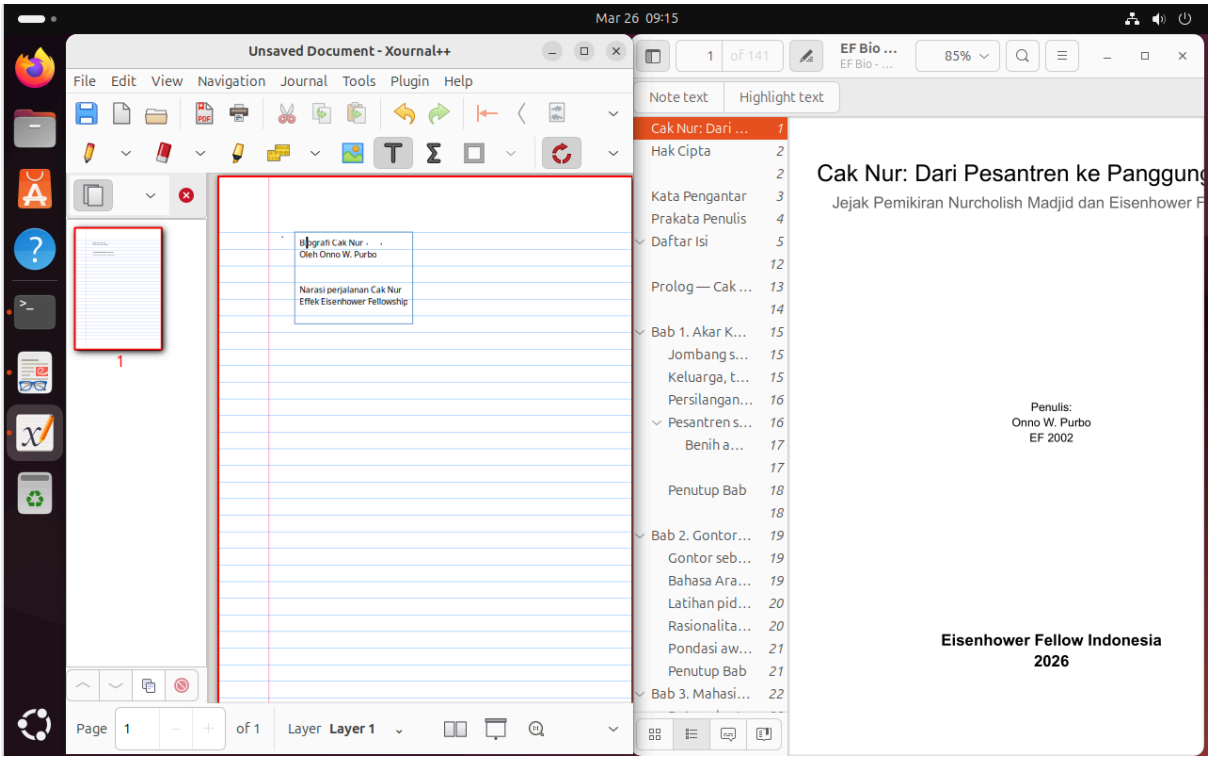


Figure. xournalpp dan PDF Viewer

Social, Messaging, and Collaboration Apps

Ubuntu is also a strong platform for social communication, messaging, and collaborative work. *Ubuntu The Complete Guide* explicitly includes sections on social networks and setting up email, which reinforces that Ubuntu is meant to participate in everyday connected life, not stand apart from it. The media and browser references also show streaming, remote collaboration, and service access through both browser and desktop apps.

For many modern users, collaboration is not optional. Group projects, online classes, remote offices, community coordination, and academic teamwork all depend on communication tools. Ubuntu supports this through a combination of native apps, browser apps, and cross-platform clients.

A practical collaboration stack may include:

- messaging clients
- video-conference tools such as **Zoom**
- browser-based collaboration suites
- file-sharing and cloud-connected tools
- screen-recording or screenshot tools for explaining problems and workflows

The Wayland and PipeWire improvements mentioned in the references are especially relevant here because they improve desktop recording, screen sharing, and audio/video behavior in modern applications. That makes Ubuntu even more practical for online teaching, collaboration, and meetings.

A helpful illustration is to think of collaboration tools as the meeting table in the room. The desktop is no longer just a solo workspace. It is also a place where ideas are exchanged.

CLI example: download the Zoom installer

```
cd ~/Downloads
wget https://zoom.us/client/latest/zoom_amd64.deb
```

Example output:

```
--2026-03-26 10:15:00--
https://zoom.us/client/latest/zoom_amd64.deb
Resolving zoom.us... done
Connecting to zoom.us... connected.
Saving to: ‘zoom_amd64.deb’
```

CLI example: install Zoom

```
sudo apt -y install gdebi libxcb-xtest0 libglx-mesa0 \
libxcb-xinerama0 libxcb-cursor0

sudo dpkg -i zoom_amd64.deb
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
 zoom
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

CLI example: launch Zoom

```
zoom &
```

Example output:

```
[1] 6210
```

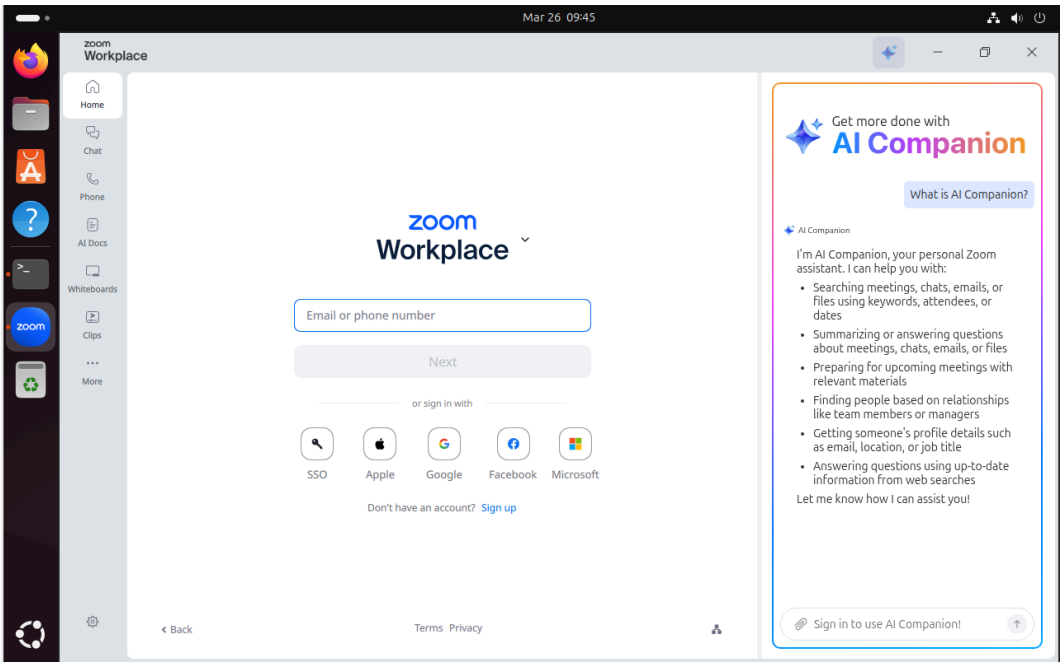


Figure. ZOOM Workspace

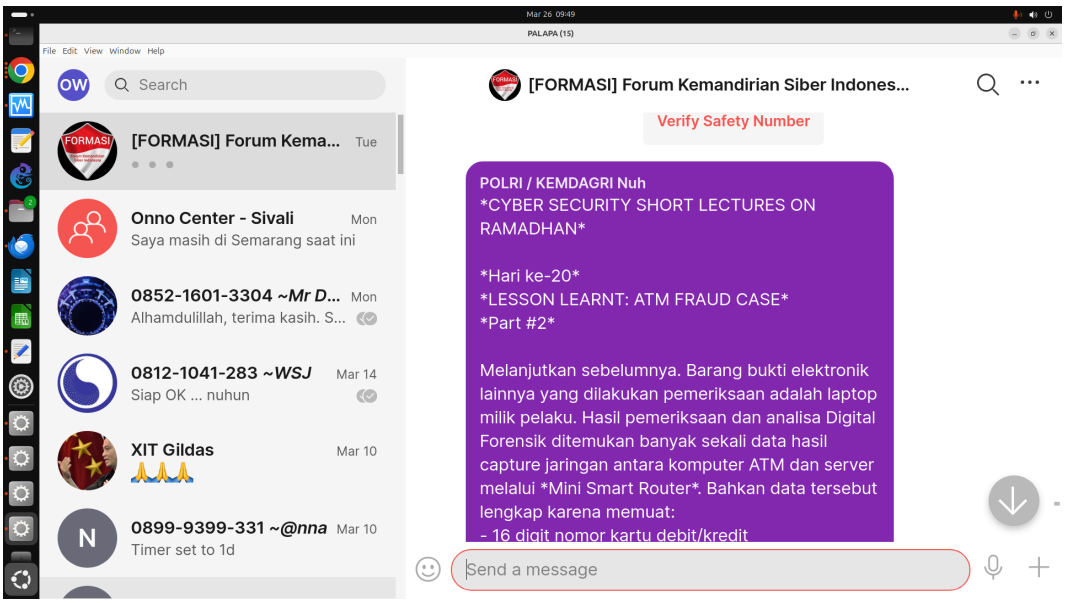


Figure. Palapa Xecure Chat made by Indonesians

App Selection Strategy: Native Package vs Snap vs Flatpak vs .deb

Choosing an application on Ubuntu is not only about *what* the app does. It is also about **how it is packaged**. The Ubuntu Academy outline explicitly emphasizes app-source choices across APT/native packages, Snap, Flatpak, and `.deb` workflows. The references explain the practical trade-offs: official repository packages are trusted and integrated, Snaps are self-contained and well integrated with Ubuntu, Flatpaks offer sandboxed cross-distribution packaging, and `.deb` packages are useful for direct vendor delivery but should be handled carefully.

A helpful decision model for students and practitioners is:

- choose a **native Ubuntu repository package** when you want strong system integration, trust, and simplicity
- choose a **Snap** when Ubuntu offers a well-maintained package and you want a self-contained application with background updates
- choose a **Flatpak** when the app is strong on Flathub or when you want a desktop-oriented sandboxed package
- choose a **local `.deb`** only when the vendor distributes it directly and you trust the source

The Ubuntu Linux Bible explains that Snap and Flatpak reduce dependency conflicts by carrying much of what they need with them and by running in more isolated environments. At the same time, Mastering Ubuntu notes that Snaps can have a larger disk footprint and longer startup time compared with traditional `.deb` packages. Those are exactly the kinds of practical trade-offs that learners should understand.

A simple illustration may help:

- **native package** = eating at the campus dining hall: familiar, integrated, and managed centrally
- **Snap** = a packaged meal box: convenient, self-contained, easy to deliver
- **Flatpak** = a travel food kit: designed to work in many environments with fewer conflicts
- **`.deb`** = food delivered directly from a specific restaurant: sometimes excellent, but you must trust the source

CLI example: compare the same idea across package methods

APT search:

```
apt search vlc
```

Example output:

```
vlc/noble 3.0.21 amd64
  multimedia player and streamer
```

Snap search:

```
snap find vlc
```

Example output:

```
Name  Version  Publisher   Notes  Summary
vlc   3.0.21   videolan✓    -      The ultimate media
player
```

Flatpak install example:

```
flatpak install flathub org.videolan.VLC
```

Example output:

```
Looking for matches…
Required runtime for org.videolan.VLC/x86_64/stable found in remote flathub
Proceed with these changes to the system installation? [Y/n]: y
```

These examples show that Ubuntu users are not trapped in one packaging path. Instead, they gain the freedom to choose the packaging model that best fits their needs.

Comparing apt, snap, flatpak and .deb

Aspect	Native Package (APT)	Snap	Flatpak	`.deb` Installer
Source	Ubuntu repository	Snap Store	Flathub / Flatpak repo	Vendor website
Install	`apt install app`	`snap install app`	`flatpak install ...`	`apt install ./app.deb`
Updates	Via `apt upgrade`	Automatic / Snap	Via `flatpak update`	Vendor-dependent
Integration	Best with Ubuntu	Good	Good	Usually good
App version	Often older	Often newer	Often newer	Often newest
Size	Smaller	Larger	Larger	Varies
Best for	Stability	Convenience	Cross-distro apps	Official vendor packages

Note: APT is best for stability, Snap and Flatpak are best for newer sandboxed apps, and `.deb` is best when the vendor provides an official installer.

Guided Everyday Apps Activity

To make this chapter practical, readers can complete a short desktop-application lab:

- open a **browser** and import or review bookmarks
- install **LibreOffice** and create a short document
- install **VLC** using either APT or Snap
- open **Rhythmbox** and explore music, radio, or podcast sections
- install **GIMP** or **Inkscape** and open a sample image
- install a **PDF/note application** and annotate a document
- install a **messaging or collaboration app**
- compare one application available as **native package**, **Snap**, or **Flatpak**

This kind of exercise helps readers move from package-management theory to real Ubuntu desktop use.

Chapter Reflection

Everyday Ubuntu Desktop applications show that Ubuntu is not only a learning platform for Linux concepts, but also a capable and enjoyable system for real work and daily life. Browsers support safe access to the web. Email and communication apps organize interaction. Office tools handle writing, tables, and presentations. Rhythmbox and VLC support music, podcasts, and video. Creative tools such as GIMP and Inkscape support visual work. PDF and note apps support reading and study. Messaging and collaboration tools keep Ubuntu connected to modern digital workflows. Finally, understanding packaging choices—native package, Snap, Flatpak, or `.deb`—helps users select not only the right app, but the right delivery method for that app.

Key Takeaways

- **Ubuntu Desktop supports all major categories of everyday applications**, from browsing to media to office work.
- **Firefox, Chromium, and Chrome-related workflows** are central to Ubuntu web use.
- **Rhythmbox and VLC** are important examples of Ubuntu media applications.
- **GIMP, Inkscape, and related creative tools** show that Ubuntu is also a visual and creative desktop.
- **Communication, collaboration, and note workflows** are fully possible on Ubuntu Desktop.
- **Package choice matters**: native packages, Snap, Flatpak, and `.deb` each have practical trade-offs.

Chapter 8 — Running Windows-Oriented Workflows on Ubuntu

Moving to Ubuntu Desktop does **not** mean abandoning all familiar workflows overnight. In real life, many students, teachers, office staff, and professionals still depend on file formats, habits, and applications shaped by Windows or macOS. That is why this chapter matters. The Ubuntu Academy outline explicitly includes **transitioning from other operating systems**, and the attached references reinforce the same idea: Ubuntu is a practical desktop system with equivalents for most common desktop tasks, while compatibility tools and virtualization can help when native Linux replacements are not enough.

A useful way to think about this chapter is to imagine moving from one office building to another. You do not carry every old wall and door with you, but you do want the same important activities to continue: writing documents, opening shared files, attending meetings, using special business tools, and finishing work on time. Ubuntu supports this transition in several layers:

- replacing many Windows-first applications with Linux-native equivalents
- preserving compatibility through common file formats
- using **Wine** for selected Windows applications
- using **virtual machines** for software that truly needs Windows
- adopting better habits for users coming from Windows or macOS

The goal is not to imitate Windows perfectly. The goal is to help readers work confidently on Ubuntu while still handling the practical reality of mixed-platform environments.

Replacing Common Windows Applications

One of the most encouraging lessons for new Ubuntu users is that **most everyday Windows tasks already have a strong Ubuntu alternative**. The attached references explicitly describe Ubuntu Desktop as including or supporting the usual categories of software people expect on Windows, such as **Firefox**, **Chrome**, **VLC**, **LibreOffice**, and **Thunderbird**. They also note that Linux desktops now have equivalent applications for almost every major type of desktop task found on Windows or Mac systems.

This is an important mindset shift. New users often ask, “Where is the Linux version of Microsoft Word?” The better question is usually, “What is the **best Ubuntu tool** for the work I need to do?” In many cases, the answer is not a clone but an equivalent.

A practical translation table looks like this:

- **Microsoft Edge / Chrome → Firefox** or **Chromium**
- **Microsoft Office → LibreOffice**
- **Windows Media Player → VLC**
- **Outlook-style desktop email workflow → Thunderbird**
- **Notepad++-style quick editing** → native Linux editors or, in some cases, Wine-based use of Notepad++

A good illustration is to think of Ubuntu as a campus in a different country. The room names may be different, but the library, classroom, office, and workshop still exist. Users do not lose the ability to work; they simply learn the local map.

CLI example: install common Windows-style equivalents

```
sudo apt update
sudo apt install libreoffice thunderbird vlc
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  libreoffice thunderbird vlc
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
```

CLI example: search for a browser alternative

```
apt search firefox
apt search chromium-browser
```

Example output:

```
firefox/noble 1:1snap1-0ubuntu2 amd64
  Mozilla Firefox web browser

chromium-browser/noble 1:124.0 amd64
  Chromium web browser
```

These examples show that many familiar desktop tasks can be handled natively on Ubuntu without extra compatibility layers.

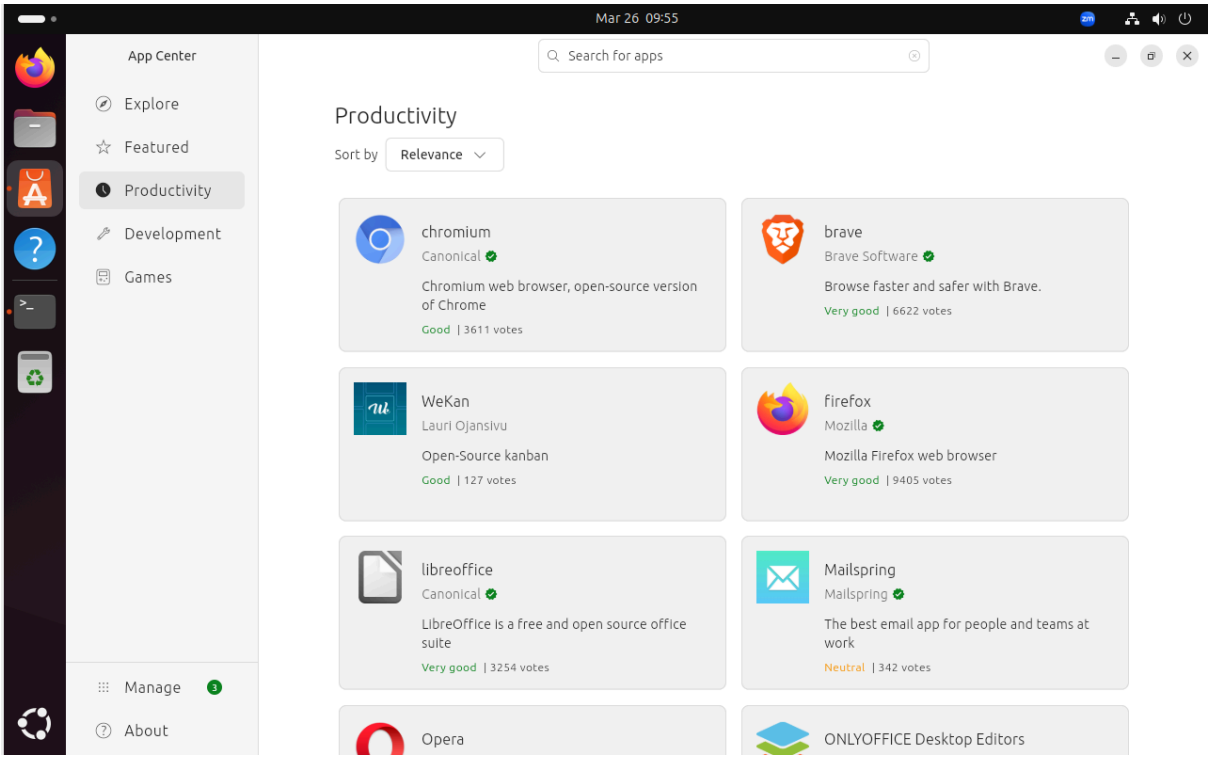


Figure. App Center - Productivity apps.

File Compatibility and Cross-Platform Formats

One of the most practical concerns for users moving from Windows or macOS is whether their files will still open properly. In most day-to-day cases, the answer is yes, especially when users rely on **common cross-platform formats** and understand a few platform differences. The attached references discuss Ubuntu as a real desktop system for documents, browsing, and productivity, and they also point out some filesystem and path differences that become relevant when moving between Linux and Windows programs.

For everyday work, readers should get comfortable with formats such as:

- **PDF** for final-form documents
- **ODT**, **ODS**, and **ODP** for open office formats
- **DOCX**, **XLSX**, and **PPTX** when collaboration with Microsoft Office users is necessary
- **PNG**, **JPG**, and **SVG** for images
- **MP3**, **MP4**, and common media formats when codecs are available
- plain text and **UTF-8** text files for notes, scripts, and exchange across systems

The Mohr reference also highlights one subtle but important migration detail: **Linux and Windows treat paths and filenames differently**. Linux uses forward slashes `/`, while Windows uses backslashes `\`. Linux is also case-sensitive, so `Report.docx` and `report.docx` are treated as different names. This matters when users move documents between systems, use scripts, or run Windows-oriented applications through Wine.

A simple illustration helps here. Think of file compatibility as speaking with someone from another region. The language is mostly shared, but accent and spelling differences still matter. Most documents travel well, but a few habits need adjustment.

CLI example: inspect a downloaded cross-platform document

```
ls ~/Documents
file ~/Documents/report.docx
```

Example output:

```
report.docx  budget.xlsx  slides.pptx
/home/student/Documents/report.docx: Microsoft Word
2007+
```

CLI example: demonstrate Linux path style

```
pwd
echo "/home/student/Documents/report.docx"
```

Example output:

```
/home/student
/home/student/Documents/report.docx
```

CLI example: show case sensitivity

```
touch Report.txt
touch report.txt
ls
```

Example output:

```
Report.txt  report.txt
```

This is a small example, but it teaches one of the most important cross-platform differences clearly.

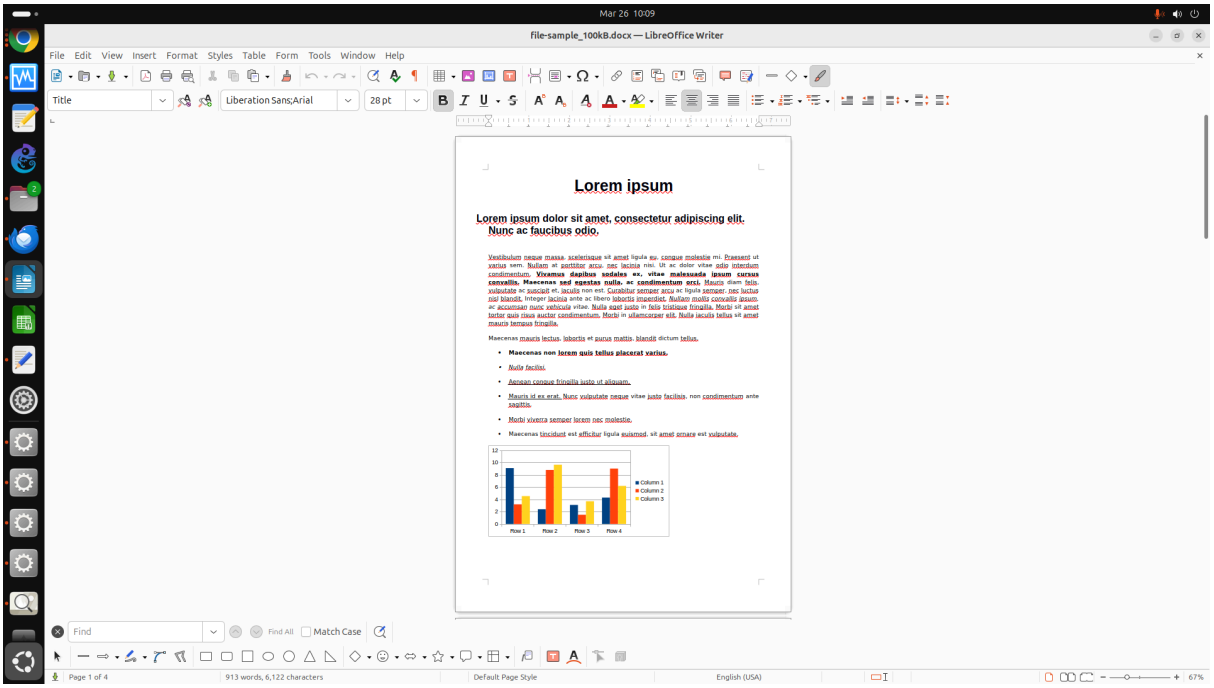


Figure. LibreOffice Word read sample .docx file

Using Wine and Compatibility Layers

Sometimes a native Ubuntu application is not enough. In those cases, users may turn to **Wine**, which the attached references describe as a **compatibility layer** that allows many Microsoft Windows programs to run on Linux. The Mohr reference explains that Wine can be used at a basic level to install and run selected Windows tools, and it also points readers to **PlayOnLinux** as a front-end offering prebuilt and tested configurations for many Windows games and applications. It further notes the existence of **CrossOver** as a commercial, business-oriented option built around Wine.

This wording matters: Wine is a **compatibility layer**, not a full Windows operating system. It tries to create an environment where a Windows application believes it is running in Windows, even though it is actually running on Ubuntu. That means compatibility can vary. Some applications work beautifully. Others work only partially. Some may not work at all.

A healthy beginner mindset is:

- use native Linux applications first when possible
- use Wine for **selected** Windows applications that are known to behave well
- test carefully before depending on Wine for critical production work
- consider **PlayOnLinux** or **CrossOver** when a guided setup is helpful

A helpful illustration is to think of Wine as an interpreter in a meeting. It can make communication possible, but the conversation may still be smoother when everyone speaks the same language natively.

CLI example: install Wine from Ubuntu repositories

```
sudo apt update
sudo apt install wine
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  wine wine64 wine32
0 upgraded, 3 newly installed, 0 to remove and 0 not
upgraded.
```

The attached guide also describes a GUI route through Ubuntu Software where Wine may appear under a name such as *Configure Wine*.

CLI example: perform Wine's first configuration

```
winecfg
```

Example output:

```
wine: created the configuration directory '/home/student/.wine'
wine: configuration in L"/home/student/.wine" has been updated.
```

This matches the reference's explanation that the first launch lets Wine build its base Windows-style folder structure.

CLI example: inspect Wine's working folder

```
ls ~/.wine
```

Example output:

```
dosdevices  drive_c  system.reg  user.reg  userdef.reg
```

The Mohr guide specifically notes that Wine stores its files under the standard `~/.wine` folder.

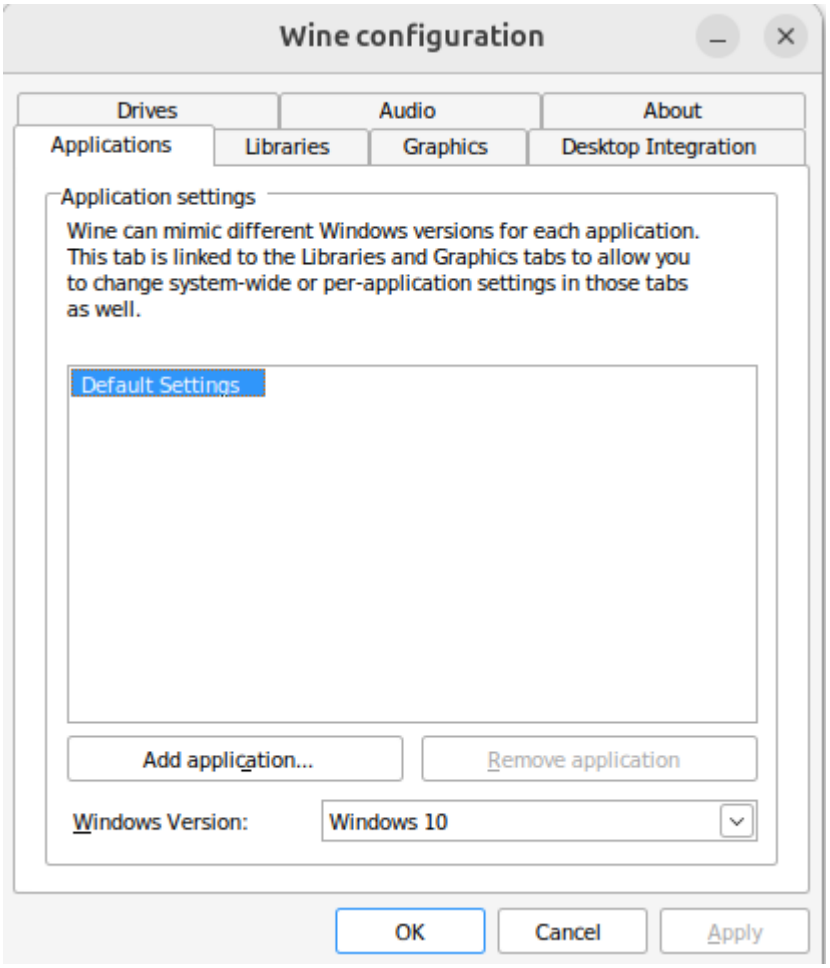


Figure. Winecfg configuration windows after first launch.

Running Selected Windows Programs

Once Wine is installed, users can try running selected Windows applications. The attached Ubuntu guide gives a concrete example using **Notepad++**: download a reputable Windows installer, right-click the `.exe`, and choose **Open in Wine** so Wine can handle the installation. The guide emphasizes that Wine must manage the installation so the application believes it is running in a Windows-like environment.

This is a practical and realistic workflow, but readers should also understand its limits. Not every Windows application is a good candidate for Wine. As the reference explains, Wine works best for known compatible applications, and front-ends like PlayOnLinux can make life easier because they provide tested setup templates.

A good rule for everyday Ubuntu use is:

- start with **simple utilities** and lightweight applications
- test installers from **trusted sources only**
- expect some applications to work better than others
- prefer native Linux software for critical work whenever possible

A nice illustration is to think of running Windows apps through Wine like using an adapter plug in a hotel room. Many devices will work through the adapter, but not every appliance is equally happy with it.

CLI example: run a Windows installer in Wine

```
wine ~/Downloads/npp.8.6.4.Installer.x64.exe
```

Example output:

```
0024:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
0024:fixme:plugplay:build_device_instance_id Unhandled BUS type 00000015
```

Warnings like these are not unusual. Some Windows applications still install and run successfully even when Wine prints non-fatal diagnostic messages.

CLI example: list Wine-installed program files

```
ls ~/.wine/drive_c/Program\ Files
```

Example output:

```
Internet Explorer  Notepad++  Windows NT
```

CLI example: launch an installed Windows application

```
wine ~/.wine/drive_c/Program\ Files/Notepad++/notepad++.exe
```

Example output:

```
(no terminal output if the GUI launches successfully)
```

The reference notes that installed Wine applications may also appear in the desktop search or have icons placed on the desktop.

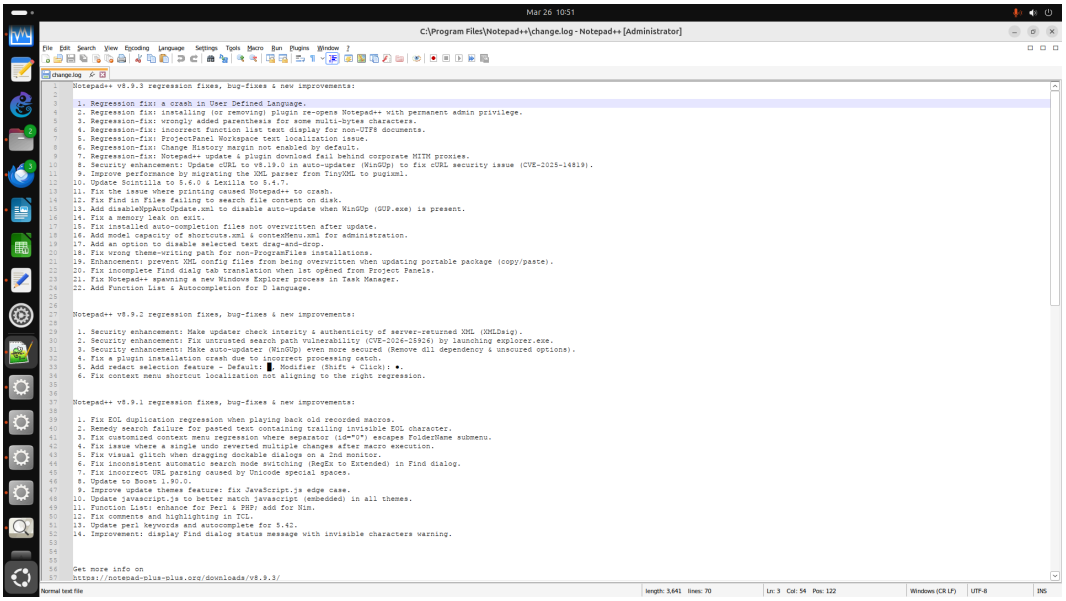


Figure. notepad++ running using wine

Virtual Machines for Legacy Software

For software that truly depends on Windows, especially older business tools or legacy educational applications, **virtual machines** can be a better choice than Wine. The attached references make this point clearly. Mohr explains that virtual machines offer an alternative where compatibility layers are only a partial solution, and that dual booting or virtualization can be valuable when applications are not available or do not work well across platforms. The Ubuntu Linux Bible also emphasizes Linux's strength as a **virtualization host**, capable of running guest operating systems such as Microsoft Windows.

A virtual machine is like setting up a fully separate computer inside a window on your Ubuntu desktop. That computer can run Windows, complete with its own applications, settings, and files. This approach often gives better compatibility than Wine because the guest software is running inside real Windows, not just inside a compatibility layer.

Virtual machines are especially useful for:

- legacy Windows-only software
- training labs and safe experiments
- short-term access to specialized business tools
- testing documents or workflows in a real Windows environment
- keeping old software available without giving up Ubuntu as the main OS

The trade-off is that virtualization needs more hardware resources: memory, storage, and CPU capacity. Still, for many users, this is the cleanest way to preserve access to old workflows.

CLI example: check whether hardware virtualization support is visible

```
lscpu | grep Virtualization
```

Example output:

```
Virtualization:      VT-x
```

CLI example: install VirtualBox on Ubuntu

```
sudo apt update
sudo apt install virtualbox
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  virtualbox
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

The Ubuntu Linux Bible specifically mentions **Oracle's VirtualBox** as one of the platforms to use when you need virtualization formats beyond LXD containers.

CLI example: list registered virtual machines

```
VBoxManage list vms
```

Example output:

```
"Windows 10 Lab" {12345678-90ab-cdef-1234-567890abcdef}
```

This command is useful for readers who want to connect GUI virtualization with command-line inspection.

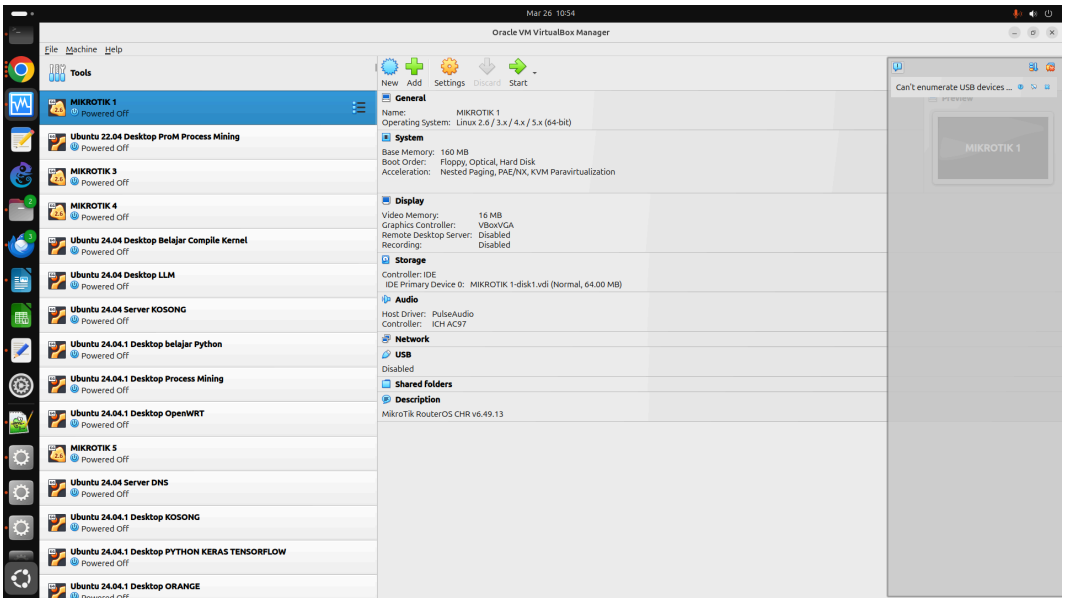


Figure. Virtualbox Manager

Migration Tips for Users Coming from Windows/macOS

Transitioning to Ubuntu is easiest when readers do not try to memorize every difference at once. The attached references strongly support a **practical migration mindset**. Ubuntu is presented as an operating system alongside Windows and macOS, but also as something broader: a distribution that bundles the desktop, browser, office tools, and many other components into one coherent environment. The references also note that people coming from Windows may either replace Windows entirely or keep both systems available through dual boot, depending on their comfort level and compatibility needs.

A good migration strategy is not “forget everything you know.” It is “map the familiar to the new.”

Useful tips include:

- **learn the equivalents**, not just the names
  - Browser → Firefox or Chromium
  - Office suite → LibreOffice
  - File Explorer → Files/Nautilus
- **expect path differences**
  - Windows: `C:\Users\Name\Documents`
  - Ubuntu: `/home/name/Documents`
- **remember case sensitivity** in Linux filenames
- **use dual boot or a VM** during the transition if one critical Windows application still exists
- **prefer open, portable file formats** where possible
- **do not force old habits unnecessarily**; let Ubuntu teach its own efficient workflow
- **keep backups** before major transitions or dual-boot changes, because partitioning always carries some risk

For Windows users specifically, the Ubuntu Cinnamon discussion in the attached beginner reference also points out that some Ubuntu-based desktops feel visually more traditional and may ease transition for users who prefer an older Windows-style layout. That does not mean GNOME is wrong; it simply means the Linux desktop world offers choices for different comfort levels.

A good illustration is to think of migration like learning to drive in another country. The car still has steering, pedals, and signs, but a few rules, road markings, and habits are different. Confidence grows quickly once the driver stops comparing every movement and starts practicing.

CLI example: show the current home directory

```
echo $HOME
pwd
```

Example output:

```
/home/student
/home/student
```

CLI example: demonstrate file path separators

```
echo "Linux path: /home/student/Documents"
echo "Windows path style uses backslashes, such as
C:\\Users\\Student\\Documents"
```

Example output:

```
Linux path: /home/student/Documents
Windows path style uses backslashes, such as
C:\Users\Student\Documents
```

CLI example: create a migration practice folder

```
mkdir -p ~/Documents/migration-practice
ls ~/Documents
```

Example output:

```
migration-practice  report.docx  budget.xlsx
```

This gives readers a simple place to practice opening familiar documents on Ubuntu.

Guided Migration Activity

To make this chapter practical, readers can complete a short migration lab:

- install **LibreOffice**, **Firefox**, **Thunderbird**, and **VLC**
- open a **DOCX**, **XLSX**, and **PDF** file on Ubuntu
- create two test files with different letter case, such as `Report.txt` and `report.txt`
- install **Wine** and run `winecfg`
- try one lightweight Windows application in Wine
- install **VirtualBox** and create a basic Windows test VM, if licensed media is available
- compare which workflow is better for a specific Windows-only application: native replacement, Wine, or VM

This type of exercise helps readers move from fear of transition to practical confidence.

Chapter Reflection

Running Windows-oriented workflows on Ubuntu is not about pretending Ubuntu is Windows. It is about understanding the layers of compatibility available. Most everyday tasks can be handled with strong native Ubuntu applications such as Firefox, LibreOffice, Thunderbird, and VLC. Common cross-platform file formats usually move well between systems, although users must remember differences such as path style and case sensitivity. When native replacements are not enough, Wine can run selected Windows applications through a compatibility layer, especially with the help of tools such as PlayOnLinux or CrossOver. For legacy or business-critical software, virtual machines often provide the cleanest solution. And for users arriving from Windows or macOS, the best migration strategy is a calm, staged transition built on equivalents, file awareness, and practical testing.

Key Takeaways

- **Most common Windows desktop tasks already have strong Ubuntu equivalents.**
- **Cross-platform file compatibility is usually good**, but users must understand Linux path and filename differences.
- **Wine is a compatibility layer**, not a full Windows replacement.
- **PlayOnLinux and CrossOver** can improve the Wine experience for some users.
- **Virtual machines are often the best answer for legacy or critical Windows-only software.**
- **Migration is easier when users map familiar tasks to Ubuntu tools instead of searching for identical clones.**

Part IV — Desktop Customization and User Experience

Chapter 9 — Customizing the GNOME Desktop

One of the most enjoyable parts of using Ubuntu Desktop is that the system can be shaped to match the user, not the other way around. The Ubuntu Academy outline explicitly includes **themes, icons, wallpapers, GNOME Extensions, GNOME Tweaks, GSettings, dconf, startup behavior, and workspace productivity** as core customization topics, which makes this chapter a practical bridge between daily comfort and desktop administration.

Customization is not only about decoration. A better wallpaper may improve comfort, but the deeper purpose of customization is to create a desktop that is **clear, efficient, and pleasant to use for long periods of time**. The attached references describe GNOME as a focused, modern desktop designed to reduce distraction, while also showing that Ubuntu users can change themes, icons, shell behavior, scaling, and extensions to make the environment more personal and more productive.

A useful illustration is to think of GNOME as a well-designed study room. The room already works when you first enter it, but after a while you may want to adjust the lighting, move the shelf, choose a different chair, organize your desk differently, and place your most-used tools within easier reach. That is what desktop customization is: **making a good workspace feel like your own workspace**.

Appearance Settings

Ubuntu's default GNOME desktop already presents a polished visual style, but the built-in **Settings** application provides the safest and simplest entry point for everyday appearance changes. The Ubuntu Academy outline places **Settings > Appearance** at the center of beginner-friendly customization, and the attached references also describe Ubuntu and GNOME as desktops where visual preferences can be changed gradually without needing to break the system.

For many users, the first appearance adjustments are small:

- changing the wallpaper
- choosing light or dark mode
- confirming icon visibility and layout preferences
- adjusting display scale on HiDPI screens
- making sure text is comfortable to read

These basic settings matter because comfort affects productivity. A student reading on a bright screen for hours may prefer dark mode. A user on a high-resolution display may need scaling. Someone moving from another operating system may simply want the desktop to feel more visually familiar.

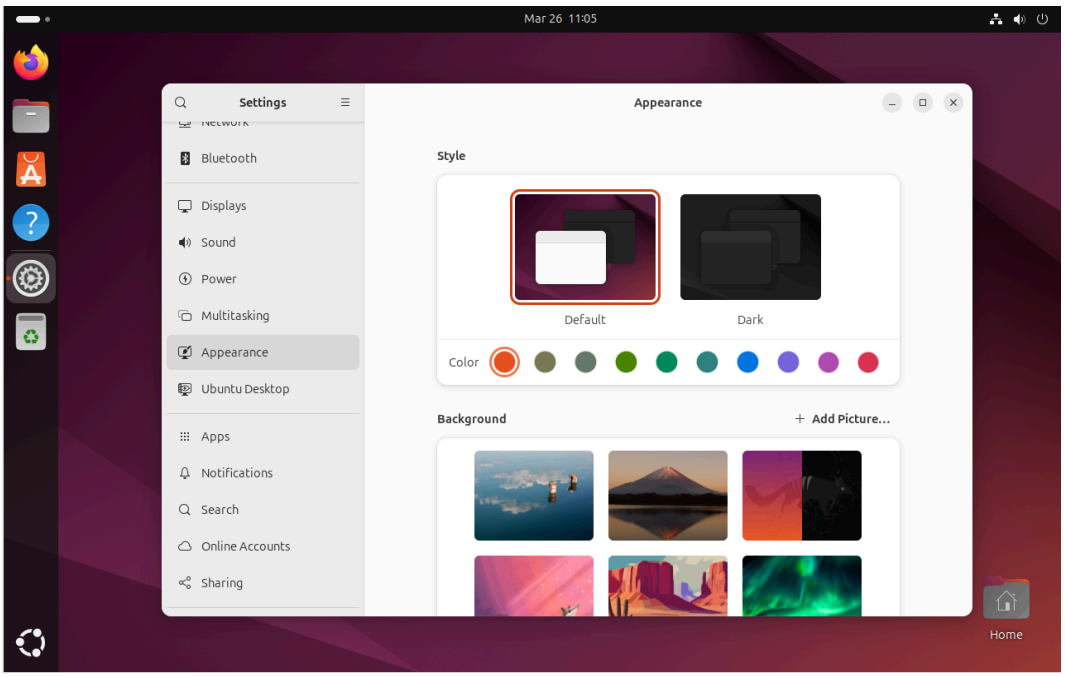


Figure. Appearance Setting in Ubuntu

Themes

Themes define much of the visual personality of the desktop. They influence how windows, menus, controls, and other interface elements look. The attached beginner reference explains that Linux allows many desktop modifications and that themes can be changed from the **Appearance** area of the tweak tool, including window and shell-related visual elements. It also notes that users can download additional themes and apply them for deeper personalization.

The Ubuntu Complete Guide also explains the broader principle behind themes very well: many graphical desktop elements are stored in a theme, which makes it easier to change the overall look and feel of the environment in terms of shape, color, and other visual characteristics.

A practical customization principle is this: **choose a theme that improves readability and comfort, not just novelty**. A theme that looks dramatic in screenshots may be tiring after hours of real use. Productive desktops usually benefit from clear contrast, moderate color choices, and a design that keeps attention on the work rather than on the interface itself.

CLI example: inspect current GTK theme

```
gsettings get org.gnome.desktop.interface gtk-theme
```

Example output:

```
'Yaru'
```

CLI example: change the GTK theme

```
gsettings set org.gnome.desktop.interface gtk-theme
'Yaru-dark'
```

Typical result:

```
(no output if successful)
```

This kind of command is useful for advanced users who want reproducible configuration steps, but beginners should still start with the graphical tools first. The reference materials explicitly link GNOME customization with GSettings and tweak tools, making this command-line bridge appropriate in an Ubuntu Academy context.

Screenshot note:
[Screenshot-CH09-03: GNOME Tweaks Appearance section showing theme choices for applications, shell, and legacy windows.]

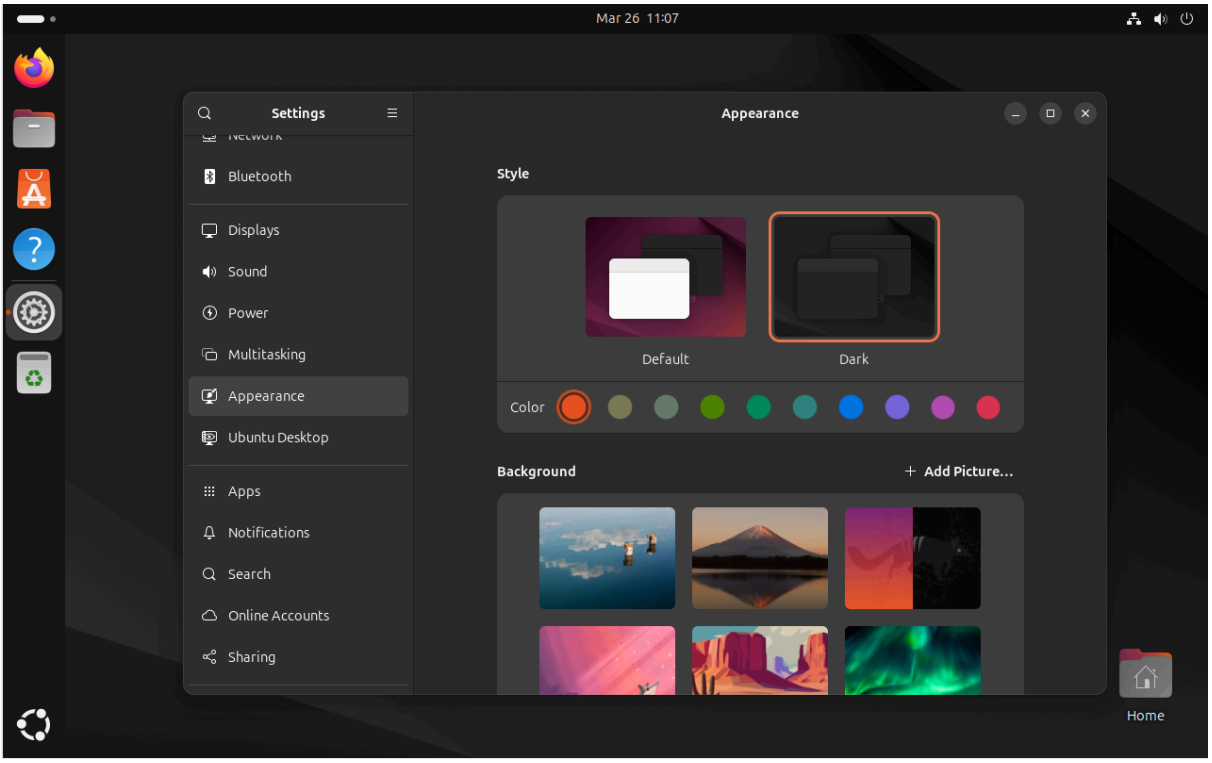


Figure. Dark Mode Theme

Icons

Icons are small, but they strongly influence how a desktop feels. A clear icon set makes applications and folders easier to recognize quickly, while a poor icon set can make the desktop feel inconsistent or harder to scan. The attached beginner reference explicitly lists **icons** as one of the items that can be changed through the tweak tool, alongside theme, cursor, background, lock screen image, shell, and sound.

Icons also matter psychologically. A desktop with coherent icons feels more “finished” and easier to trust. This is especially relevant in Ubuntu, where GTK-based applications, system tools, and third-party apps may come from different packaging sources. A consistent icon theme can help unify the overall look.

CLI example: inspect the current icon theme

```
gsettings get org.gnome.desktop.interface icon-theme
```

Example output:

```
'Yaru'
```

CLI example: set a different icon theme

```
gsettings set org.gnome.desktop.interface icon-theme
'Yaru'
```

Typical result:

```
(no output if successful)
```

Even when the chosen value is unchanged, this command teaches an important principle: icon sets are part of the desktop interface configuration, not just an artistic extra.

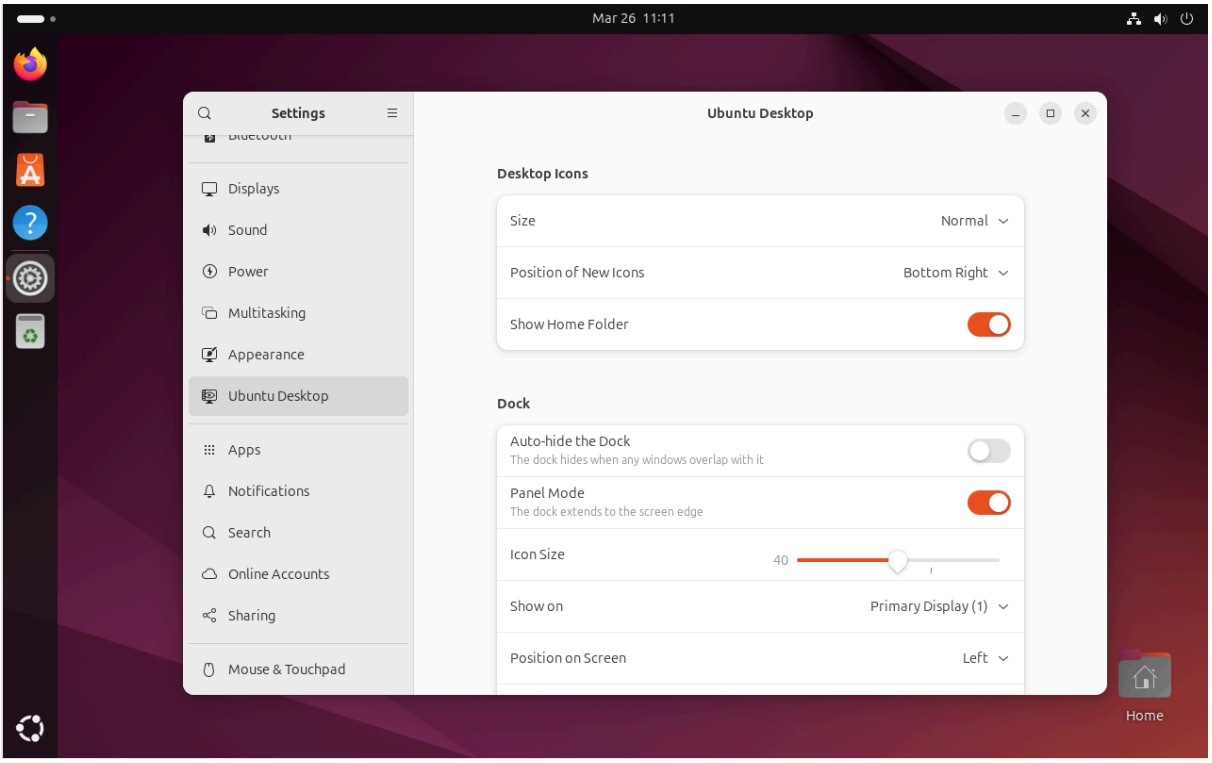


Figure. Icon Settings in Ubuntu Desktop.

Wallpapers

Wallpapers are often the first thing users change because they are easy, visual, and immediately personal. The Ubuntu Linux Bible exercise material even uses changing the background as a practical GNOME desktop task: download an image and set it as the current background. That shows how central wallpaper customization is in basic desktop familiarity.

A wallpaper should do more than look nice. A good wallpaper supports the rest of the desktop. If it is too busy, it competes with icons and windows. If it is too bright, it may create eye strain. If it is too dark, text or UI contrast may feel unbalanced depending on the rest of the theme.

CLI example: set a wallpaper with GSettings

```
gsettings set org.gnome.desktop.background picture-uri
'file:///home/student/Pictures/wallpaper.jpg'
```

Typical result:

```
(no output if successful)
```

CLI example: inspect the current wallpaper

```
gsettings get org.gnome.desktop.background picture-uri
```

Example output:

```
'file:///home/student/Pictures/wallpaper.jpg'
```

These commands help advanced readers understand that even very visual changes can be automated or documented, which is useful in labs, training rooms, or multi-machine desktop setups.

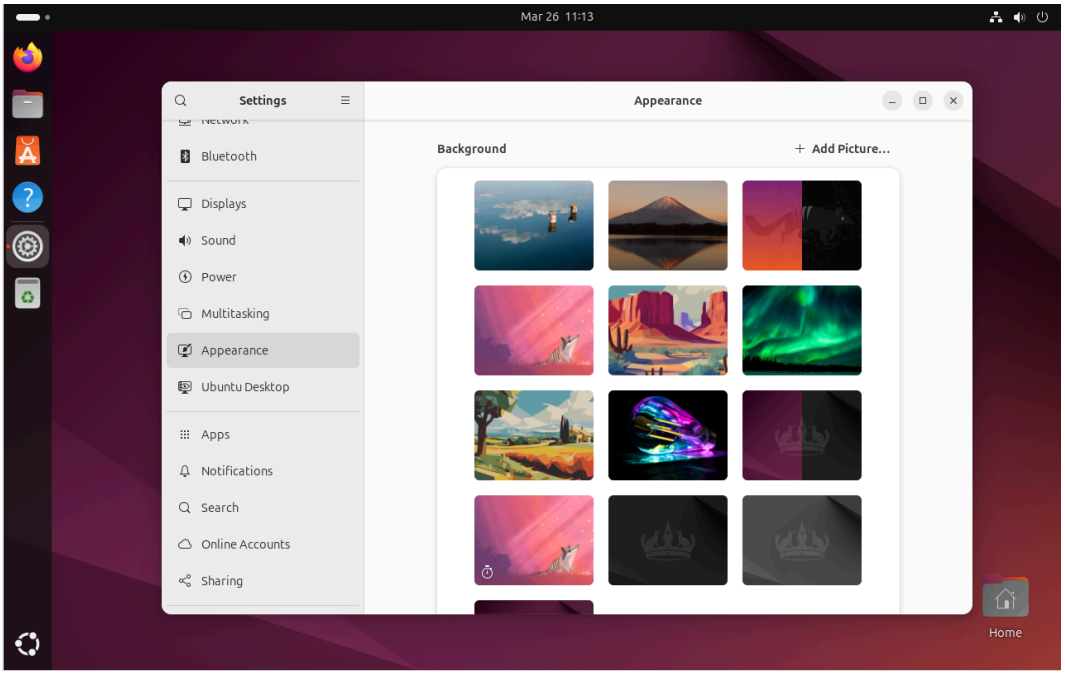


Figure. Wallpaper setting.

Dark Mode

The Ubuntu Complete Guide highlights one of the most visible modern desktop improvements: Ubuntu now offers a **dark mode**, described as easier on the eyes for many users. It also specifically connects dark mode with the **Yaru** theme and the more mature, polished modern Ubuntu desktop experience.

Dark mode is not just a trend. For some users, especially those working at night or in low-light rooms, it improves comfort. For others, a light interface remains easier to read during long daytime sessions. The important lesson is not that dark mode is always better, but that **Ubuntu lets users choose the visual balance that best matches their eyes and working conditions**.

CLI example: prefer dark mode

```
gsettings set org.gnome.desktop.interface color-scheme
'prefer-dark'
```

Typical result:

```
(no output if successful)
```

CLI example: inspect current color-scheme preference

```
gsettings get org.gnome.desktop.interface color-scheme
```

Example output:

```
'prefer-dark'
```

This is a nice example of how modern Ubuntu desktop customization combines simple graphical controls with scriptable system settings.

Dock and Panel Customization

GNOME and Ubuntu place strong emphasis on a clean top bar and a side Dock, but users often want to adapt these areas to their own workflow. The Ubuntu Academy outline explicitly includes **dock/panel adjustments**, while the attached references describe the Dock, top bar, system menu, and the way GNOME organizes windows and applications around them.

The Ubuntu Linux Bible explains that new applications add icons to the Dock and that right-clicking applications can pin them or open new windows. The beginner GNOME reference also describes the top bar and system menu as key parts of GNOME Shell. Together, these sources support an important customization point: the Dock and top bar are not only decorative; they are central workflow tools.

Useful Dock customizations may include:

- pinning favorite applications
- removing rarely used icons
- changing size or behavior through tweaks or extensions
- arranging work so that the Dock supports common tasks instead of becoming cluttered

A good desktop Dock is like a tool rack beside your desk. The tools you use every day should be close by. The ones you rarely touch should not crowd the space.

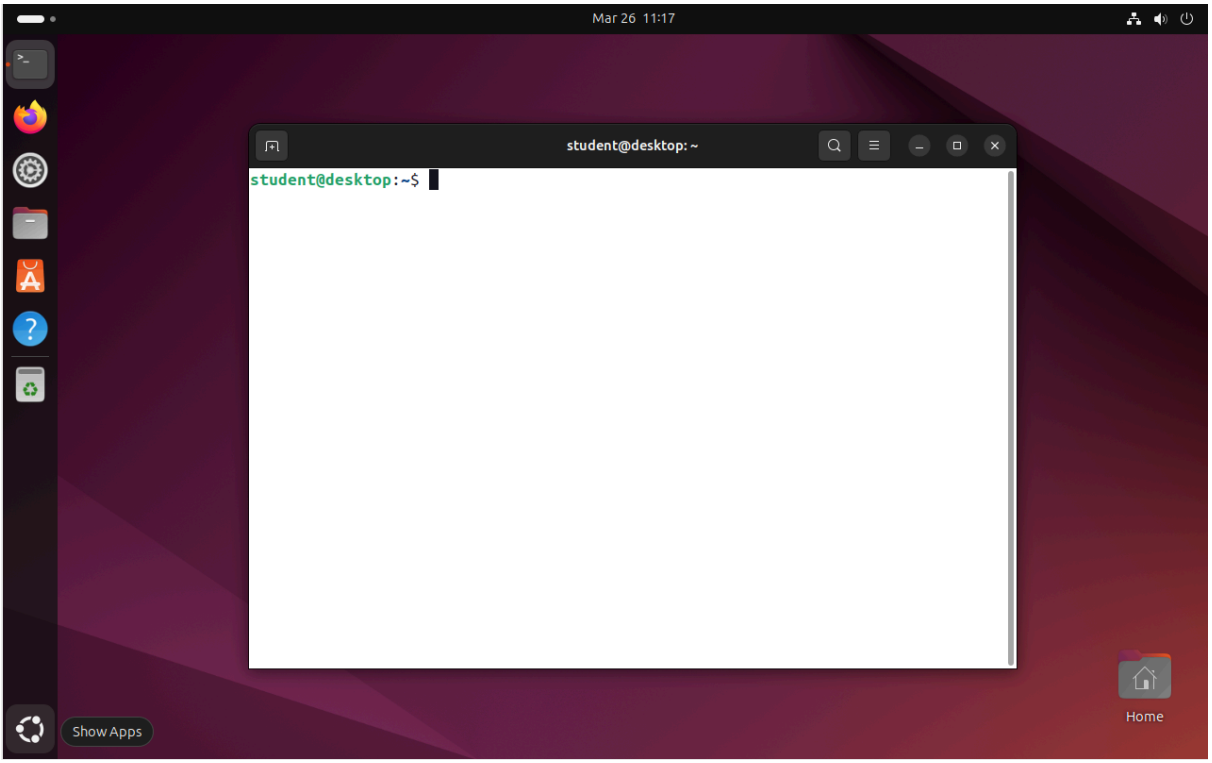


Figure. Dock with pinned Terminal and Settings

Fonts, Cursor, and Scaling

Text clarity and pointer comfort matter more than many users expect. The attached beginner reference explicitly notes that the tweak tool can change **cursor**, **icons**, and other appearance elements. The Ubuntu Complete Guide also highlights **fractional scaling** for HiDPI displays, explaining that Ubuntu now allows more comfortable scaling through the graphical display settings rather than requiring command-line GSettings work. It also warns that fractional scaling may involve trade-offs such as power usage and slight sharpness changes.

This is an important professional lesson: a productive desktop is not always the one with the most effects. It is the one where the text is readable, the cursor is easy to track, and the screen scale feels natural on the actual hardware being used.

CLI example: inspect interface font

```
gsettings get org.gnome.desktop.interface font-name
```

Example output:

```
'Ubuntu 11'
```

CLI example: inspect cursor theme

```
gsettings get org.gnome.desktop.interface cursor-theme
```

Example output:

```
'Yaru'
```

CLI example: check current scaling factor

```
gsettings get org.gnome.desktop.interface scaling-factor
```

Example output:

```
uint32 1
```

On some systems, scale and fractional scaling settings may be handled primarily through graphical display settings rather than only through these direct values. That matches the reference’s point that newer Ubuntu versions made HiDPI adjustment easier through Settings.

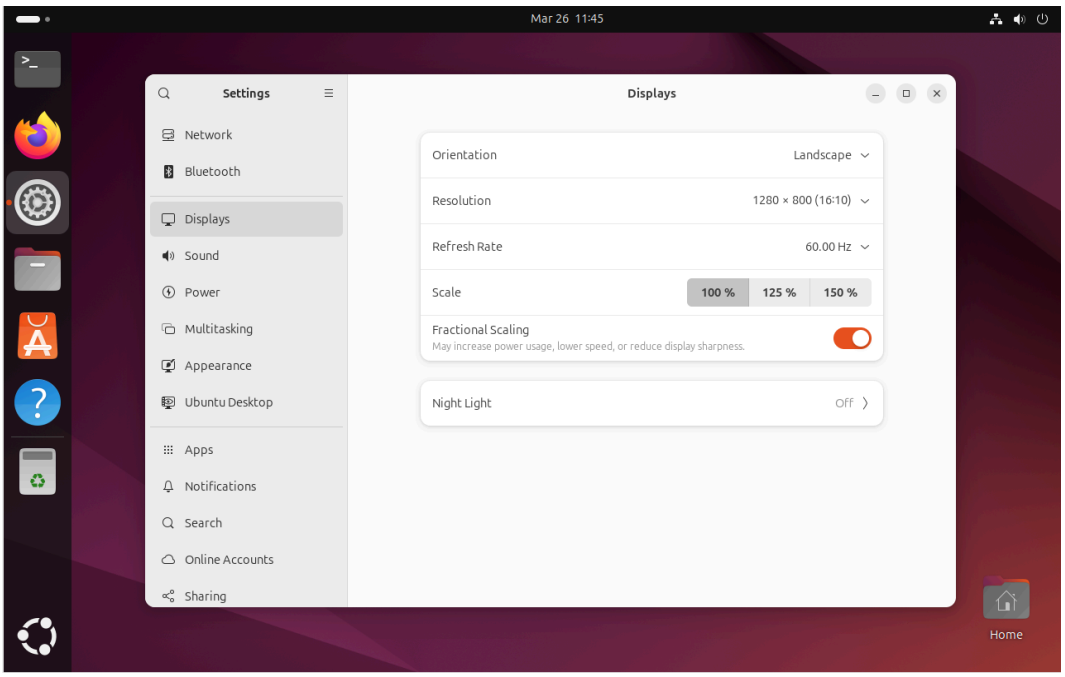


Figure. Display settings showing scaling options

GNOME Extensions

GNOME Extensions are one of the most powerful ways to customize the desktop. The attached beginner reference directly explains that GNOME Shell Extensions customize the default Shell interface and its components, including **window management** and **application launching**. It also explains that extensions can be installed per user or machine-wide, and that specific GSettings keys govern shell extension behavior.

In practical terms, GNOME Extensions let users add or modify desktop features without replacing the entire desktop environment. Common extension use cases include:

- changing how windows are managed
- adding indicators or menus
- improving tiling or workspace flow
- modifying the Dock or top bar
- restoring familiar behaviors for users coming from other environments

This matters because Ubuntu Desktop can remain GNOME at its core while still adapting to different working styles.

A useful way to think of extensions is as modular add-ons for your study room. You are not rebuilding the room; you are adding specific shelves, hooks, or tools that improve how you work.

CLI example: list enabled GNOME extensions

```
gnome-extensions list --enabled
```

Example output:

```
ubuntu-dock@ubuntu.com
ding@rastersoft.com
```

CLI example: list all installed GNOME extensions

```
gnome-extensions list
```

Example output:

```
ubuntu-dock@ubuntu.com
ding@rastersoft.com
user-theme@gnome-shell-extensions.gcampax.github.com
```

The extension-management concept is strongly supported by both the Ubuntu Academy outline and the GNOME Shell reference material.

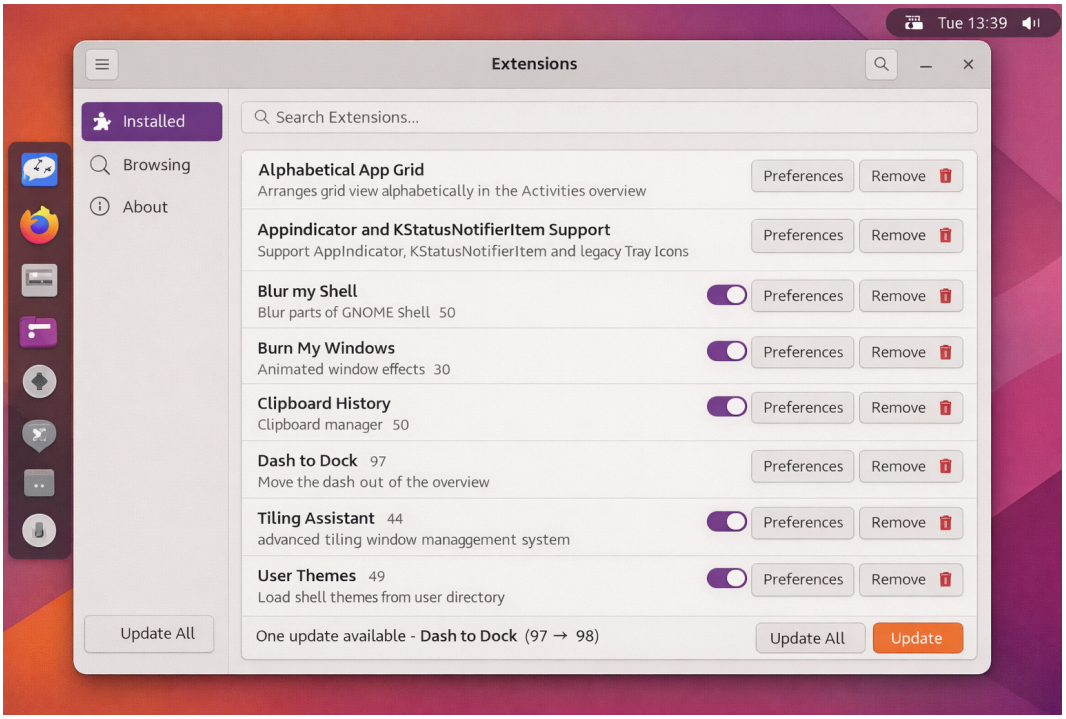


Figure. GNOME Extensions

GNOME Tweaks

The references make it clear that when the built-in Settings app feels too limited, a **tweak tool** becomes the next level of desktop control. The beginner guide explicitly shows a **GNOME Tweak** window and states that themes can be changed from the Appearance tab. It also mentions that animations can be disabled to save battery, which is a good reminder that customization can improve not only appearance but also system behavior.

GNOME Tweaks is valuable because it collects useful desktop controls in one place. It often provides access to settings that are more specific than the regular Settings app, including fonts, shell-related adjustments, appearance, startup-related behavior, and extension interaction.

CLI example: install GNOME Tweaks

```
sudo apt update
sudo apt install gnome-tweaks
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  gnome-tweaks
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

CLI example: launch GNOME Tweaks

```
gnome-tweaks &
```

Example output:

```
[1] 6412
```

A practical suggestion for readers is to treat GNOME Tweaks as a workshop panel: useful and powerful, but best approached methodically. Change one thing, observe the result, and avoid making too many interface changes at once.

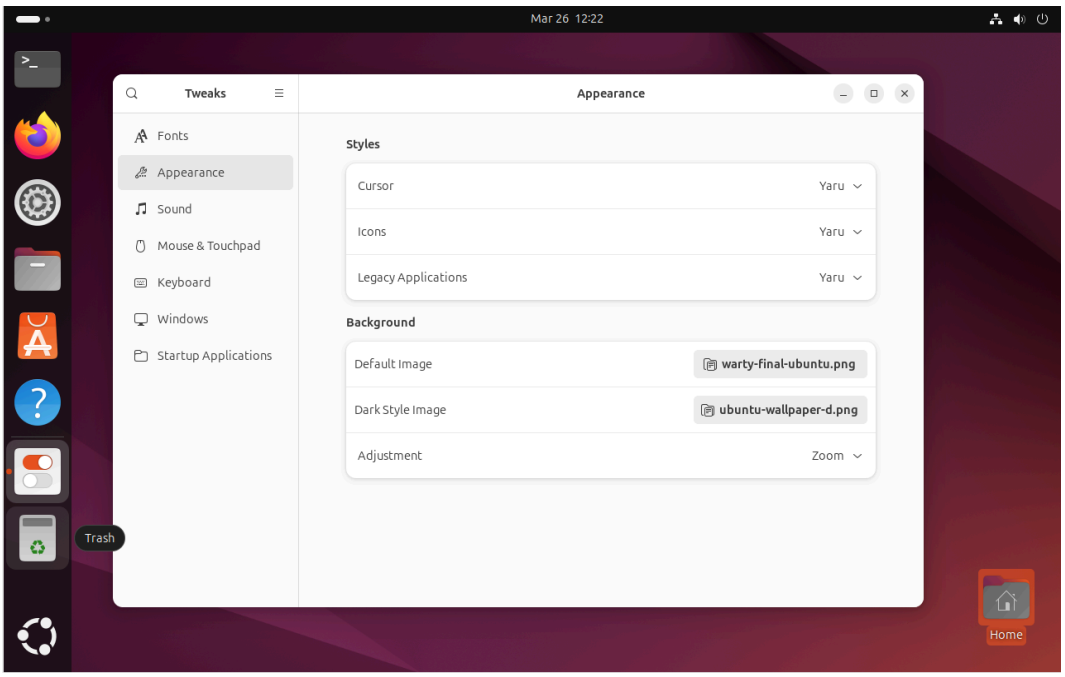


Figure. GNOME Tweaks

Advanced Tweaks with GSettings and dconf

For users who want more precision or automation, **GSettings** and **dconf** offer deeper control over GNOME behavior. The attached GNOME reference explicitly identifies **GSettings** as the storage system replacing older configuration methods, and it also links GNOME Shell Extensions to specific GSettings keys. The Ubuntu Academy outline similarly includes **GSettings / dconf for advanced tweaks**, which makes them a natural fit for this chapter.

These tools matter because they let advanced users:

- inspect desktop settings precisely
- script repeatable desktop configurations
- troubleshoot hidden preferences
- manage settings across multiple machines or accounts in training labs

However, deeper access also means deeper responsibility. A good rule is simple: **use graphical tools first when they are sufficient, and use GSettings or dconf when you need precision, documentation, or repeatability**.

CLI example: read a GNOME preference with GSettings

```
gsettings get org.gnome.desktop.interface
clock-show-weekday
```

Example output:

```
false
```

CLI example: enable weekday display in the top bar clock

```
gsettings set org.gnome.desktop.interface
clock-show-weekday true
```

Typical result:

```
(no output if successful)
```

CLI example: inspect a dconf path

```
dconf dump /org/gnome/desktop/interface/
```

Example output:

```
[/]
clock-show-weekday=true
color-scheme='prefer-dark'
gtk-theme='Yaru-dark'
icon-theme='Yaru'
```

This style of configuration inspection is especially useful when documenting a preferred desktop layout for classrooms, labs, or reproducible setups.

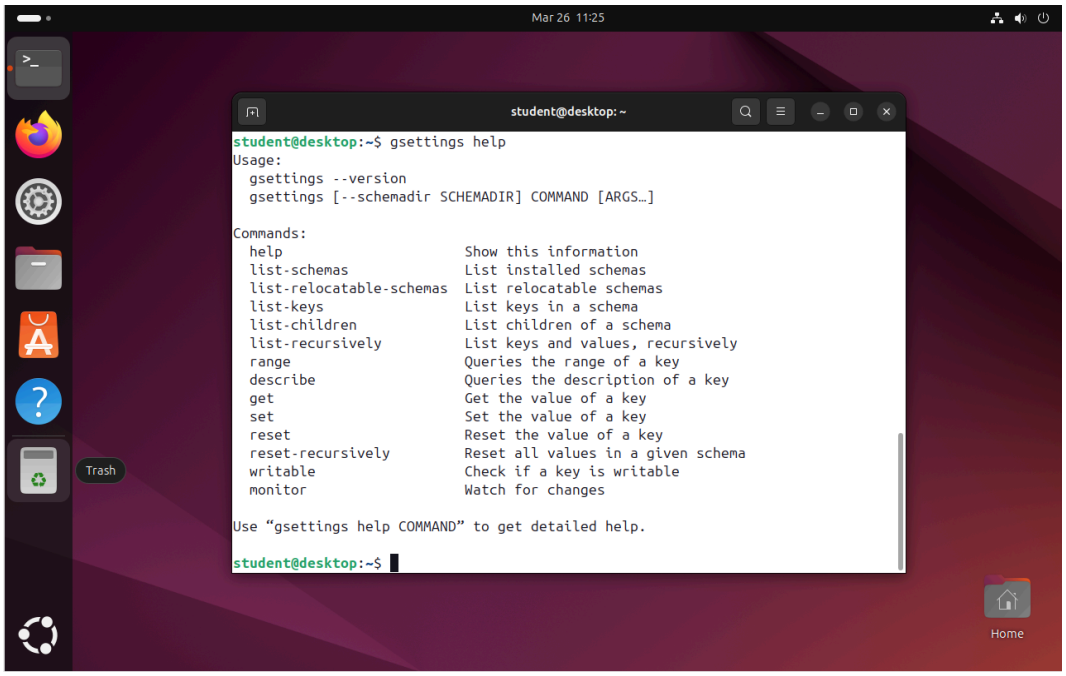


Figure. gsettings command

Desktop Icons, Startup Apps, and Session Behavior

Some users prefer a very clean desktop with no icons, while others want quick-access items such as Home, Trash, or mounted devices visible immediately. The tweak-oriented reference from the Ubuntu guide highlights this exact idea in the desktop-customization context by describing control over **desktop icons**, including items such as the Home Folder, Network, Rubbish Bin, and connected device icons. Even though that example comes from an older desktop context, the principle remains relevant: users differ in how much visual desktop access they want.

The Ubuntu Academy outline also explicitly includes **managing startup applications and sessions**, which is more than a cosmetic topic. Startup behavior affects speed, focus, and system tidiness. Too many programs loading automatically can make the desktop feel slow and cluttered. Too few can make routine work less convenient.

Practical session-behavior adjustments may include:

- deciding whether desktop icons are shown
- choosing which apps start automatically
- controlling animations or effects for battery life
- deciding how the session behaves after login
- managing extension load behavior for reliability and performance

A good analogy is this: startup applications are like the lights and appliances that turn on when you enter your office. Some are essential. Too much waste power and distracts you.

CLI example: inspect autostart entries

```
ls ~/.config/autostart
```

Example output:

```
telegram-desktop.desktop
update-notifier.desktop
```

CLI example: disable a startup application by moving its autostart file

```
mkdir -p ~/.config/autostart-disabled
mv ~/.config/autostart/telegram-desktop.desktop
~/.config/autostart-disabled/
```

Typical result:

```
(no output if successful)
```

CLI example: check the current desktop session

```
echo $XDG_CURRENT_DESKTOP
echo $DESKTOP_SESSION
```

Example output:

```
ubuntu:GNOME
ubuntu
```

This helps users connect visible session behavior to the underlying desktop environment identity.

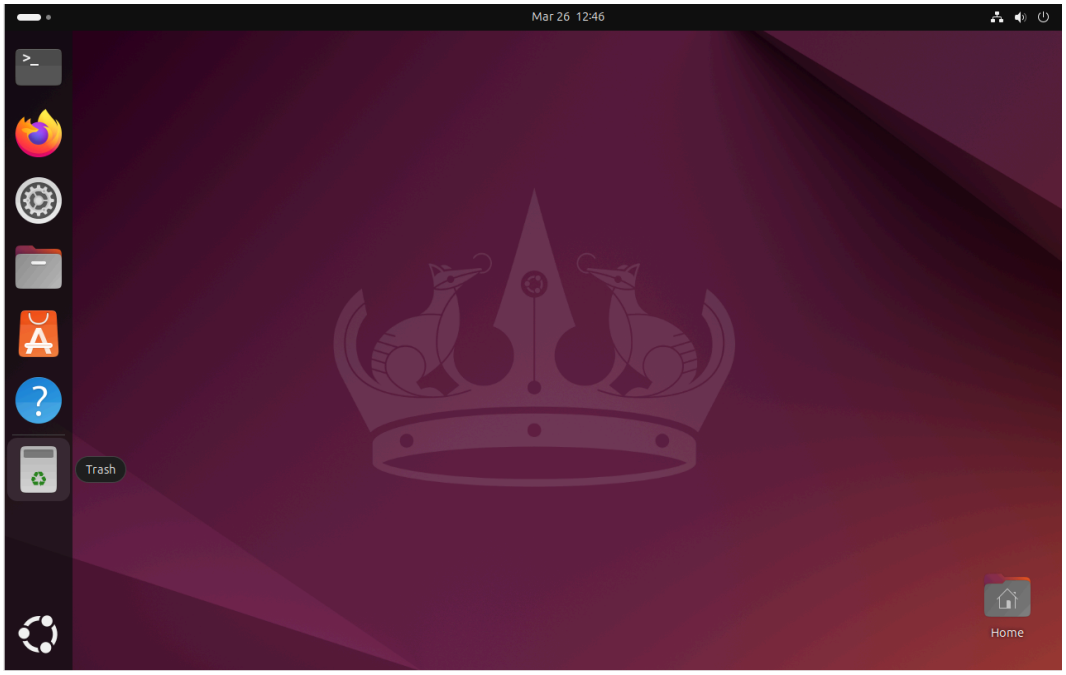


Figure. Desktop showing Home Icon enabled

Designing a Clean, Productive Desktop Workspace

The final goal of customization is not maximum modification. It is **a clean and productive workspace**. The attached references repeatedly describe GNOME as a focused environment that minimizes distraction and supports getting things done. They also emphasize workspaces, application launching, the Dock, and the possibility of choosing lighter or more feature-rich environments depending on hardware and workflow.

A productive Ubuntu desktop usually benefits from a few simple principles:

- keep only frequently used apps pinned to the Dock
- choose a wallpaper that does not compete with your work
- use readable fonts and comfortable scaling
- install only extensions that solve a real problem
- avoid over-customizing until the desktop becomes unstable or distracting
- reduce startup clutter
- use workspaces intentionally
- keep visual style consistent across themes, icons, and cursor choices

A beautiful desktop is nice. A **usable** desktop is better. A desktop that is both beautiful and usable is ideal.

A final illustration may help: imagine a writing desk. If every pen, book, cable, and decoration is left out at once, the desk becomes stressful. If the desk is too empty, it may be impractical. The best workspace contains **just enough tools, placed intentionally**. GNOME customization should follow that same philosophy.

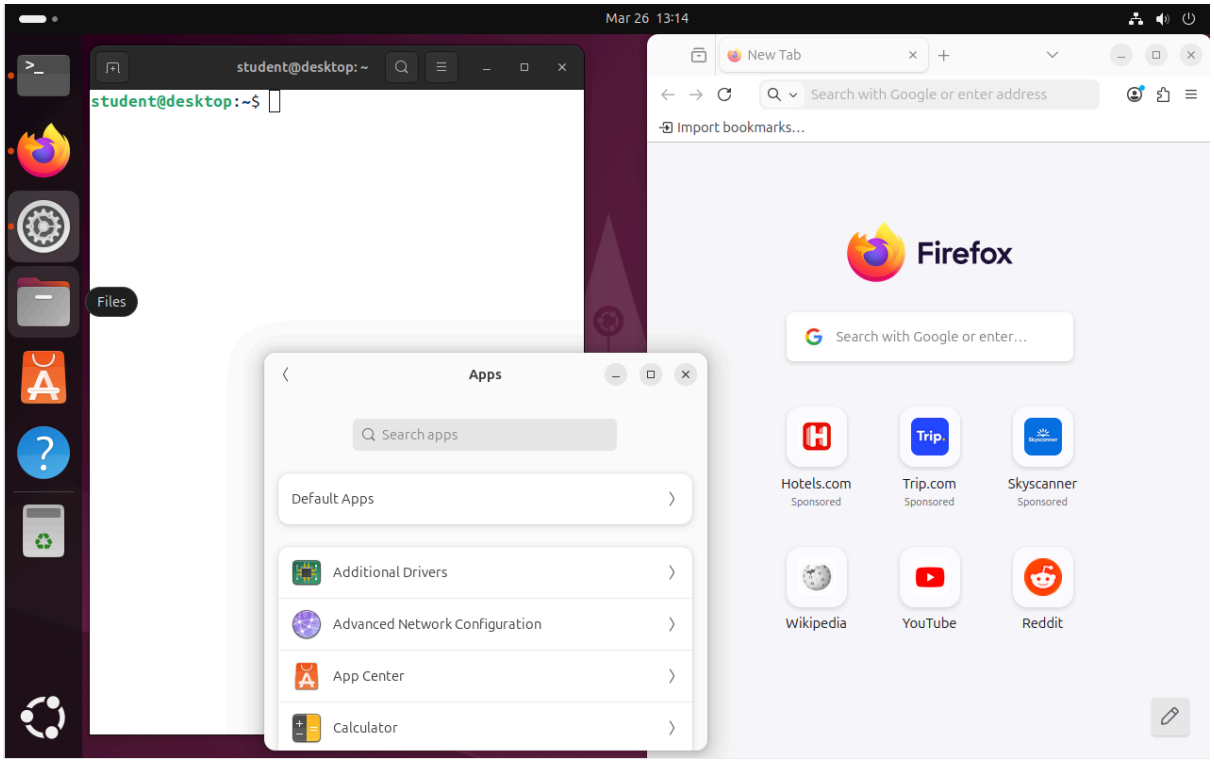


Figure. Multi-Workspace

Guided Customization Activity

Readers can turn this chapter into a practical lab:

- change the wallpaper from **Settings > Appearance**
- switch between **light** and **dark** mode
- install **GNOME Tweaks**
- inspect the current theme with `gsettings get org.gnome.desktop.interface gtk-theme`
- enable one useful **GNOME Extension**
- change an icon or cursor theme using Tweaks
- adjust scaling if using a HiDPI display
- check startup apps under `~/.config/autostart`
- use `dconf dump /org/gnome/desktop/interface/` to record current appearance settings
- redesign the Dock so only the most useful applications remain pinned

This kind of exercise helps readers see customization as a controlled, professional skill rather than random decoration.

Chapter Reflection

Customizing the GNOME desktop on Ubuntu is about building a workspace that is visually comfortable, functionally efficient, and personally meaningful. Ubuntu's built-in Appearance settings provide a safe starting point for wallpapers, style choices, and basic visual changes. GNOME Tweaks opens more detailed control over themes, icons, cursor behavior, and interface preferences. GNOME Extensions let users reshape shell behavior modularly, while GSettings and dconf provide deeper, scriptable access for precise adjustments and reproducible setups. Startup behavior, desktop icons, scaling, and session choices further influence how the desktop feels every day. The best result is not the most dramatic desktop, but the one that helps the user stay focused, comfortable, and productive.

Key Takeaways

- **Customization is about productivity and comfort, not only visual style.**
- **Settings > Appearance** is the safest starting point for everyday users.
- **GNOME Tweaks** provides deeper control over themes, icons, cursor, and behavior.
- **GNOME Extensions** modify shell behavior in modular ways and can greatly improve workflow.
- **GSettings and dconf** are useful for advanced, repeatable, and scriptable desktop tuning.
- **Scaling, fonts, and dark mode** matter because readability and comfort affect daily work.
- **A clean desktop is usually a better desktop** than an overloaded one.

Chapter 10 — Files, Search, and Daily Workspace Management

A desktop becomes truly useful when users can **find, organize, move, protect, and share files confidently**. In the Ubuntu Academy outline, everyday desktop operations explicitly include **file search, screenshots and recording, workflow optimization, and practical desktop use**, which makes this chapter a natural continuation of the GNOME desktop and customization chapters.

The attached Ubuntu references also make it clear that file work is central to desktop life. They describe the default file manager as a practical bridge between the Linux filesystem and daily user tasks, show that Ubuntu's launcher and file manager are key navigation tools, and highlight search, bookmarks, file transfer, removable devices, and desktop sharing-related improvements as part of the everyday Ubuntu experience.

A useful illustration is to think of Ubuntu Desktop as a well-organized office. The applications are your tools, but the **files are your actual work**. If your documents are lost, scattered, poorly named, or hard to recover, even a beautiful desktop becomes frustrating. Good file management turns Ubuntu from "an installed OS" into **a dependable workspace**.

Using Files (Nautilus) Efficiently

On Ubuntu Desktop, the default graphical file manager is **Files**, commonly known by its historic name **Nautilus**. The attached references describe it as the main way users navigate files and directories visually, especially in desktop use. They also note that the default file explorer is simple to use and suitable for normal file and directory navigation, while the Ubuntu Linux Bible explicitly frames Nautilus as the standard file manager in GNOME desktop exercises.

Using Files efficiently means more than opening folders. It means learning how to treat the file manager as a **control center** for everyday work:

- open and organize documents
- browse the **Home** directory
- manage **Downloads**, **Documents**, **Pictures**, and other common folders
- access mounted USB drives
- drag and drop files between locations
- move quickly between favorite places and recent locations

A beginner-friendly mental model is this: Files is like the main cabinet and hallway map of your workspace. Once you understand it, the rest of the desktop becomes easier to manage.

CLI example: open the current folder in Terminal

```
pwd
```

Example output:

```
/home/student
```

This shows the current directory in text form. It helps readers connect what they see in Files with the real Linux path underneath.

CLI example: list the contents of the Home directory

```
ls ~
```

Example output:

```
Desktop  Documents  Downloads  Music  Pictures  Public
Templates  Videos
```

This mirrors the same folders visible in the Files application and reinforces the connection between graphical and command-line navigation.

CLI example: launch Nautilus from Terminal

```
nautilus &
```

Example output:

```
[1] 4124
```

This demonstrates that the graphical file manager is still an application that can be launched from the shell.

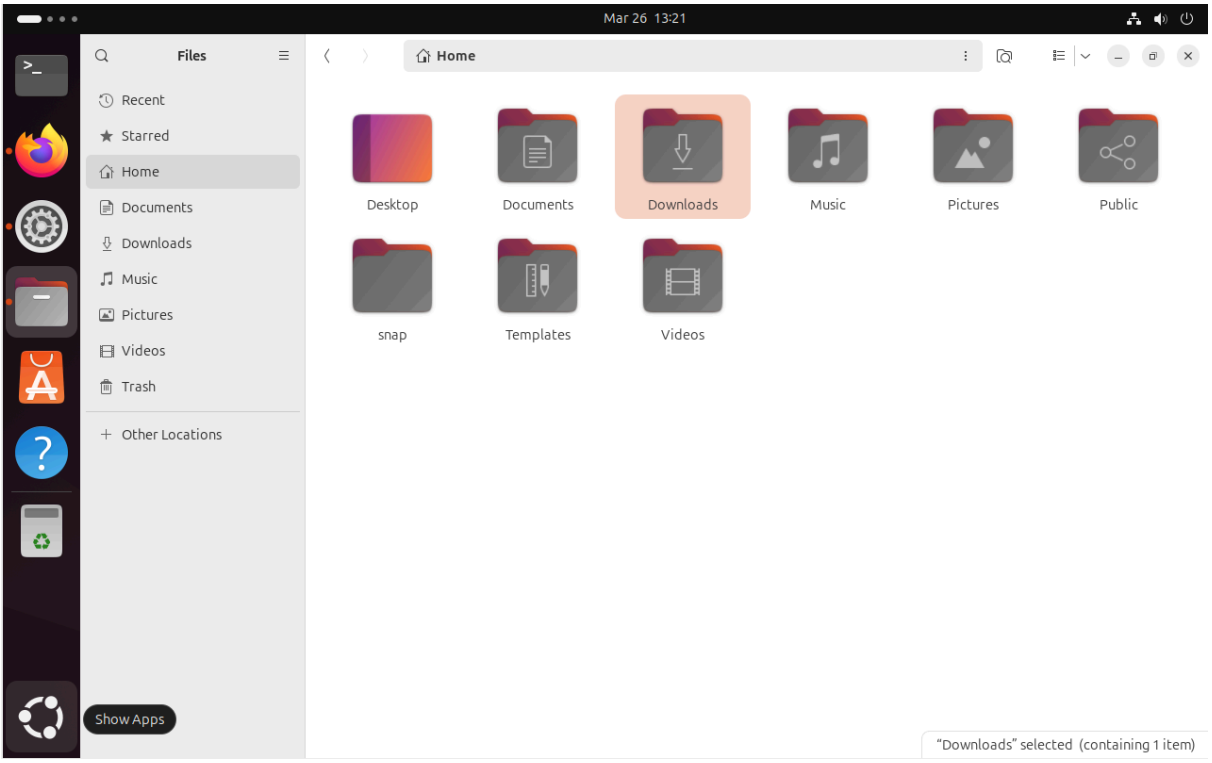


Figure. Nautilus opens in the Home Folder.

Bookmarks, Tabs, and File Operations

A file manager becomes much more powerful when users stop treating it like a single window and start using it as a navigation workspace. The references describe file managers with **shortcut panes**, location navigation, and desktop-side integration with folders and devices, all of which support the idea of working more efficiently through saved locations and structured movement.

In practical Ubuntu use, efficient file management often depends on a few habits:

- add frequently used folders to **Bookmarks**
- use multiple **tabs** or windows to compare locations
- drag and drop only when the destination is clear
- rename files immediately instead of leaving vague names like `Document1`
- create a clear folder structure early rather than reorganizing everything later

Bookmarks are especially helpful for repetitive work. A lecturer may bookmark a teaching folder. A student may bookmark research, assignments, and project folders. A media creator may bookmark video, image, and export locations.

A simple illustration is to imagine bookmarks as labels on the drawers you open every day. You do not want to walk through the whole office just to reach the same folder again and again.

CLI example: create a structured workspace folder tree

```
mkdir -p
~/Documents/Ubuntu-Academy/{Drafts,Images,References,Exp
orts}
tree -d ~/Documents/Ubuntu-Academy
```

Example output:

```
/home/student/Documents/Ubuntu-Academy
├── Drafts
├── Exports
├── Images
└── References
```

This gives readers a concrete model for a better project layout.

CLI example: copy and move files

```
cp ~/Downloads/chapter1-notes.txt
~/Documents/Ubuntu-Academy/Drafts/
mv ~/Downloads/diagram.png
~/Documents/Ubuntu-Academy/Images/
ls ~/Documents/Ubuntu-Academy/Drafts
ls ~/Documents/Ubuntu-Academy/Images
```

Example output:

```
chapter1-notes.txt
diagram.png
```

These commands reflect exactly the same actions users often perform through drag and drop in Files.

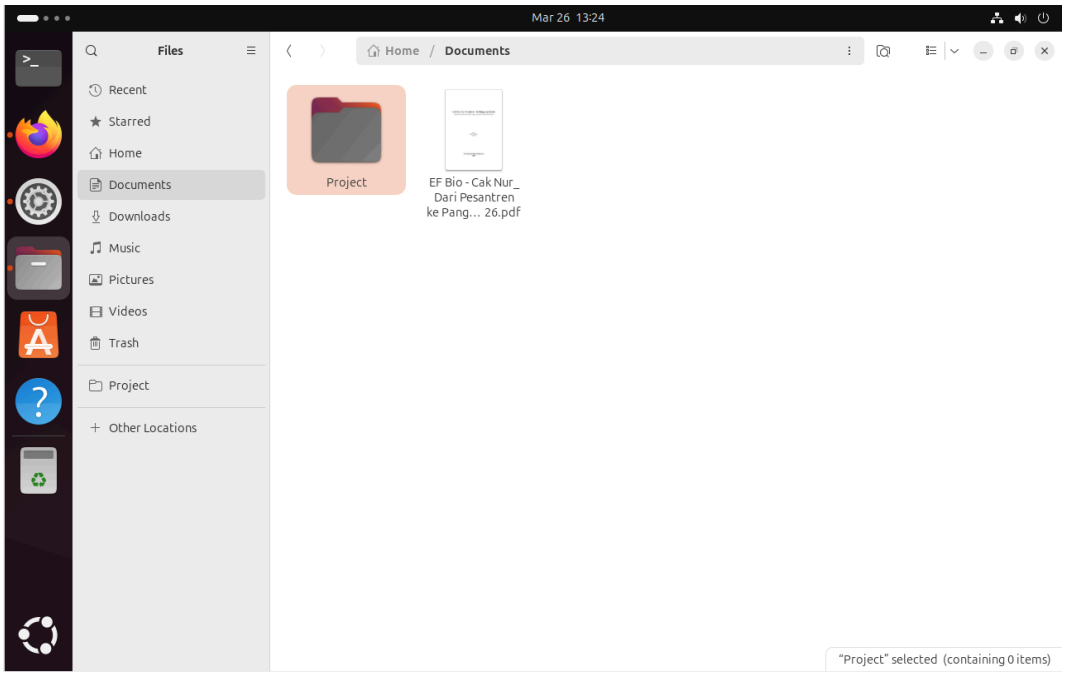


Figure. Files window with a bookmarked project folder in the sidebar

Search and Recent Files

Finding files quickly is one of the most important desktop skills. The Ubuntu Academy outline explicitly calls out **file search** as part of workflow optimization, and the Ubuntu Linux Bible provides two excellent command-line tools that reinforce the same idea: **plocate** for fast indexed searching and **find** for live, attribute-based searching. The Bible explains that `plocate` finds text within file paths very quickly, while `find` searches the filesystem directly and can filter by many attributes such as name, ownership, size, and time.

On the graphical side, Files and GNOME search help users find documents, images, and folders without memorizing every location. On the practical side, Ubuntu users should learn both GUI and CLI search methods, because each is useful in different situations.

A simple way to understand the difference is this:

- **GUI search** is like asking the receptionist to help you find a document quickly
- **`plocate`** is like checking a prepared index cabinet
- **`find`** is like walking through the building and checking the rooms directly

CLI example: fast search with plocate

```
plocate report
```

Example output:

```
/home/student/Documents/Ubuntu-Academy/Drafts/report.docx
/home/student/Documents/Ubuntu-Academy/Exports/final-report.pdf
```

The Ubuntu Linux Bible notes that `plocate` searches for the string anywhere in the file path and is generally much faster than a live search.

CLI example: case-insensitive search

```
plocate -i wallpaper
```

Example output:

```
/home/student/Pictures/Wallpapers/forest-wallpaper.jpg
/home/student/Pictures/wallpaper.jpg
```

The same reference explains that adding `-i` broadens the match by ignoring case differences.

CLI example: live search with find

```
find ~/Documents -name "*.pdf"
```

Example output:

```
/home/student/Documents/Ubuntu-Academy/Exports/final-report.pdf
/home/student/Documents/References/linux-guide.pdf
```

CLI example: search with detailed listing

```
find ~/Documents -name "*.pdf" -ls
```

Example output:

```
1298845    812 -rw-rw-r--   1 student student 829440 Mar 14 08:20
/home/student/Documents/Ubuntu-Academy/Exports/final-report.pdf
```

The Ubuntu Linux Bible explicitly highlights `find` with `-ls` as a way to print ownership, permission, size, and related details for matched files.

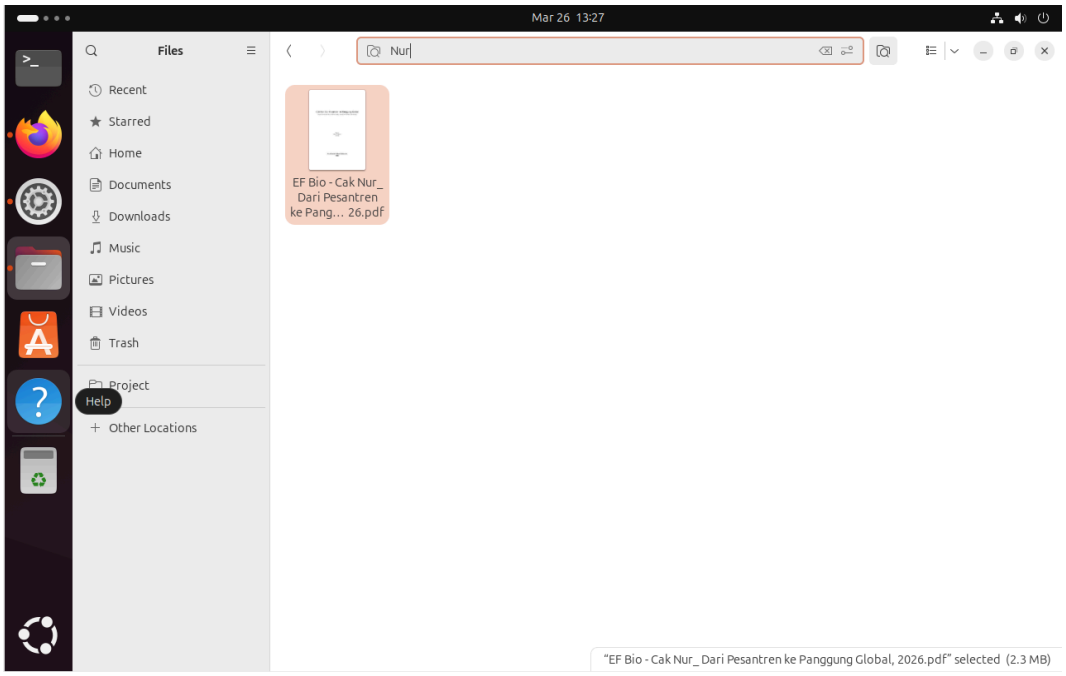


Figure. File Search bar being used.

Archives and Compression

Compressed archives are part of everyday desktop life. Users receive ZIP files by email, download compressed source packages, store grouped folders as archives, and share bundles of documents through compressed formats. Even when the attached references do not treat archives as a standalone chapter, the broader Ubuntu desktop workflow clearly supports common file packaging tasks through GUI tools and CLI commands, which fit naturally within desktop administration and file handling.

A compressed archive is like a suitcase. Instead of carrying five separate folders one by one, you place them in one container that is easier to move or send. Compression may also reduce size, especially for text-heavy materials.

Ubuntu desktop users should be comfortable with:

- opening `.zip`, `.tar.gz`, and other archive types
- extracting archives to a safe target folder
- creating an archive before sharing a project
- understanding that compression does **not** replace backup

CLI example: create a ZIP archive

```
zip -r ubuntu-academy-drafts.zip
~/Documents/Ubuntu-Academy/Drafts
```

Example output:

```
 adding: home/student/Documents/Ubuntu-Academy/Drafts/ (stored 0%)
  adding: home/student/Documents/Ubuntu-Academy/Drafts/chapter1-notes.txt
(deflated 42%)
```

CLI example: create a tar.gz archive

```
tar -czvf ubuntu-academy-images.tar.gz
~/Documents/Ubuntu-Academy/Images
```

Example output:

```
/home/student/Documents/Ubuntu-Academy/Images/
/home/student/Documents/Ubuntu-Academy/Images/diagram.png
```

CLI example: extract a ZIP archive

```
unzip ubuntu-academy-drafts.zip -d
~/Documents/Recovered-Drafts
```

Example output:

```
Archive:  ubuntu-academy-drafts.zip
   creating:
/home/student/Documents/Recovered-Drafts/home/student/Do
cuments/Ubuntu-Academy/Drafts/
  inflating:
/home/student/Documents/Recovered-Drafts/home/student/Do
cuments/Ubuntu-Academy/Drafts/chapter1-notes.txt
```

These examples help readers connect the graphical “Extract Here” and “Compress…” actions to the command line.

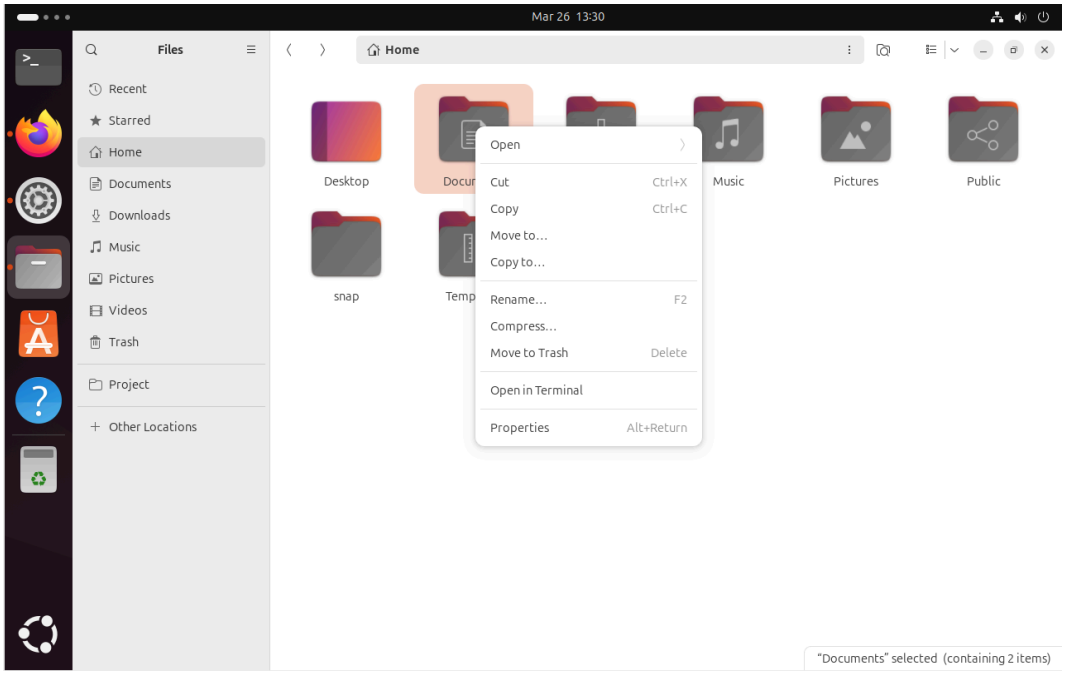


Figure. Compress Menu will archive files / folders into a .zip file.

External Drives and File Transfers

Ubuntu Desktop is designed to work smoothly with **USB drives, external disks, memory cards, and other removable media**. The attached references note that device icons or shortcuts appear when removable devices are connected and that file managers can automatically detect media such as USB sticks and optical drives. They also describe mounted devices appearing in the launcher or file-manager sidebar, which is a core part of Ubuntu's desktop usability.

This matters because file transfer is one of the most ordinary and most important tasks on a desktop system. Users copy slides to a USB drive, move media from a camera, retrieve assignments from external storage, or back up notes to removable devices.

A practical transfer workflow should include:

- confirming the correct device before copying
- opening the drive in Files
- copying files rather than dragging blindly
- waiting for write activity to finish
- **safely ejecting** the device before unplugging

A good illustration is to compare an external drive to a guest's temporary desk. You can place work there, but you should not leave in a hurry before making sure everything has been packed correctly.

CLI example: identify attached storage devices

```
lsblk
```

Example output:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 238.5G  0 disk
├─sda1   8:1    0   512M  0 part /boot/efi
└─sda2   8:2    0   238G  0 part /
sdb      8:16   1  29.3G  0 disk
└─sdb1   8:17   1  29.3G  0 part /media/student/USB32
```

CLI example: copy a file to an external drive

```
cp ~/Documents/Ubuntu-Academy/Exports/final-report.pdf
/media/student/USB32/
ls /media/student/USB32
```

Example output:

```
final-report.pdf
```

CLI example: safely unmount a removable drive

```
udisksctl unmount -b /dev/sdb1
```

Example output:

```
Unmounted /dev/sdb1.
```

The Ubuntu Academy outline also explicitly calls out **USB mounting and automount options** as part of external device management, so these commands fit directly into the intended companion-book scope.

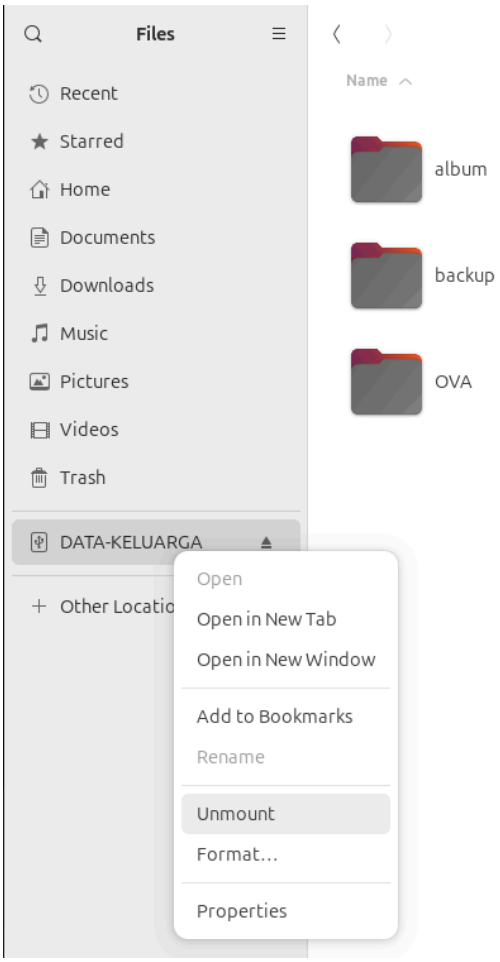


Figure. Context menu showing “Unmount” in attached USB Folder.

Trash, Recovery, and Safe Deletion

Deleting files is easy. Deleting the **right** files, and only when intended, is the real skill. The attached Ubuntu beginner reference explicitly explains that the **Trash** icon in the launcher stores files that are thrown away until the user empties the Trash permanently. It also notes that right-clicking Trash allows the user to empty it. This is an essential desktop safety feature because it creates a recovery step between “I do not need this now” and “this is gone forever.”

A healthy desktop habit is to think in three stages:

- **move to Trash** when uncertain
- **restore** if a mistake is discovered
- **empty Trash** only when confident the files are no longer needed

This is much safer than permanent deletion in a hurry.

A simple analogy helps: Trash is like the paper bin beside your desk. Throwing something there does not always mean it is immediately destroyed. It may still be retrievable until the bin is emptied.

CLI example: move a file to Trash using the command line

```
gio trash
~/Documents/Ubuntu-Academy/Drafts/old-notes.txt
```

Typical result:

```
(no output if successful)
```

CLI example: inspect the user Trash location

```
ls ~/.local/share/Trash/files
```

Example output:

```
old-notes.txt
```

CLI example: permanently remove a file

```
rm ~/Documents/Ubuntu-Academy/Drafts/temp-copy.txt
```

Typical result:

```
(no output if successful)
```

This is why `rm` should be used carefully: unlike the graphical “Move to Trash” action, it generally removes the file immediately.

CLI example: confirm a file is gone

```
ls ~/Documents/Ubuntu-Academy/Drafts
```

Example output:

```
chapter1-notes.txt
```

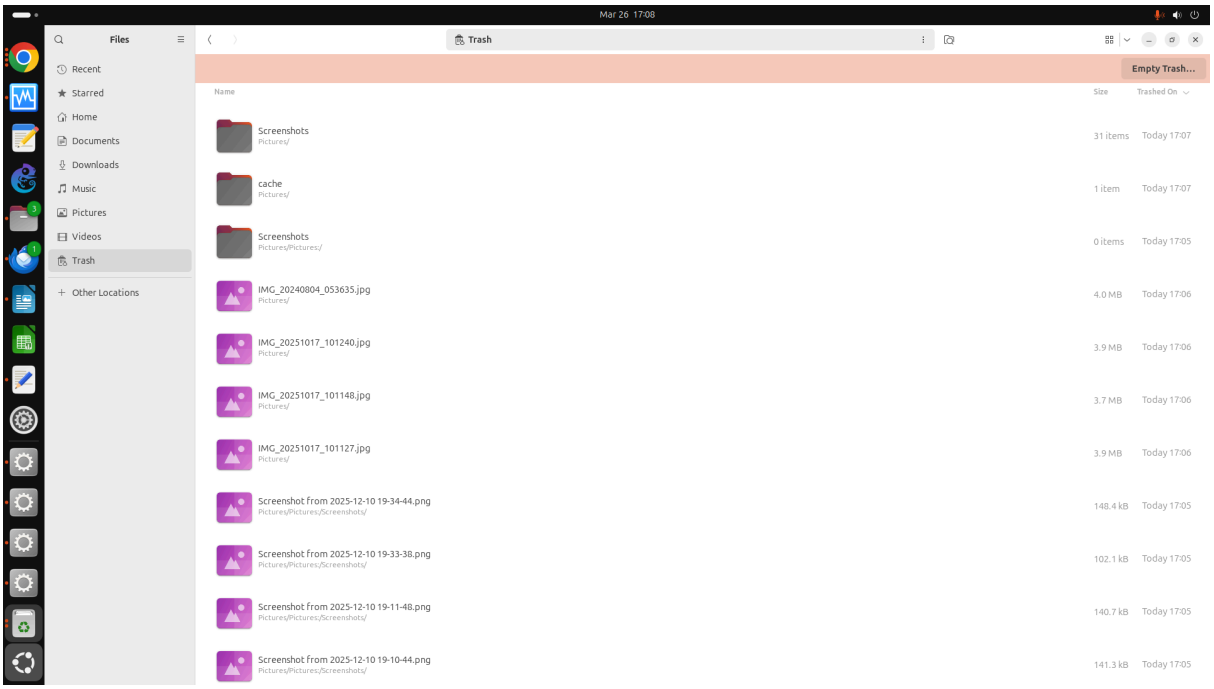


Figure. Trash window showing a file that can be restored

Screenshots, Screen Recording, and Sharing Files

Modern desktop work often includes explaining what is happening on the screen. The Ubuntu Academy outline explicitly includes **screenshots/recording** as part of everyday desktop operations, and the attached Ubuntu Complete Guide explains that newer Ubuntu desktop improvements restored and improved screen sharing and desktop recording behavior, especially through the modern Wayland and PipeWire ecosystem.

Screenshots and recordings are valuable for:

- reporting bugs
- preparing tutorials
- documenting configuration steps
- sharing results with teachers, teams, or support communities
- creating learning materials for Ubuntu itself

A screenshot is like taking a quick photo of your desk at the exact moment you want to explain something. A screen recording is like inviting someone to watch your workflow instead of just describing it.

CLI example: make sure the Pictures/Screenshots folder exists

```
mkdir -p ~/Pictures/Screenshots
ls ~/Pictures
```

Example output:

```
Screenshots  wallpaper.jpg
```

CLI example: inspect recently created screenshots

```
ls ~/Pictures/Screenshots
```

Example output:

```
Screenshot from 2026-03-14 09-12-31.png
Screenshot from 2026-03-14 09-13-04.png
```

CLI example: share a file by copying it to a common exchange folder

```
mkdir -p ~/Public/Share
cp ~/Pictures/Screenshots/*.png ~/Public/Share/
ls ~/Public/Share
```

Example output:

```
Screenshot from 2026-03-14 09-12-31.png
Screenshot from 2026-03-14 09-13-04.png
```

This reinforces a good habit: keep shared screenshots and exported files in a clearly labeled location instead of scattering them across Downloads or the Desktop.

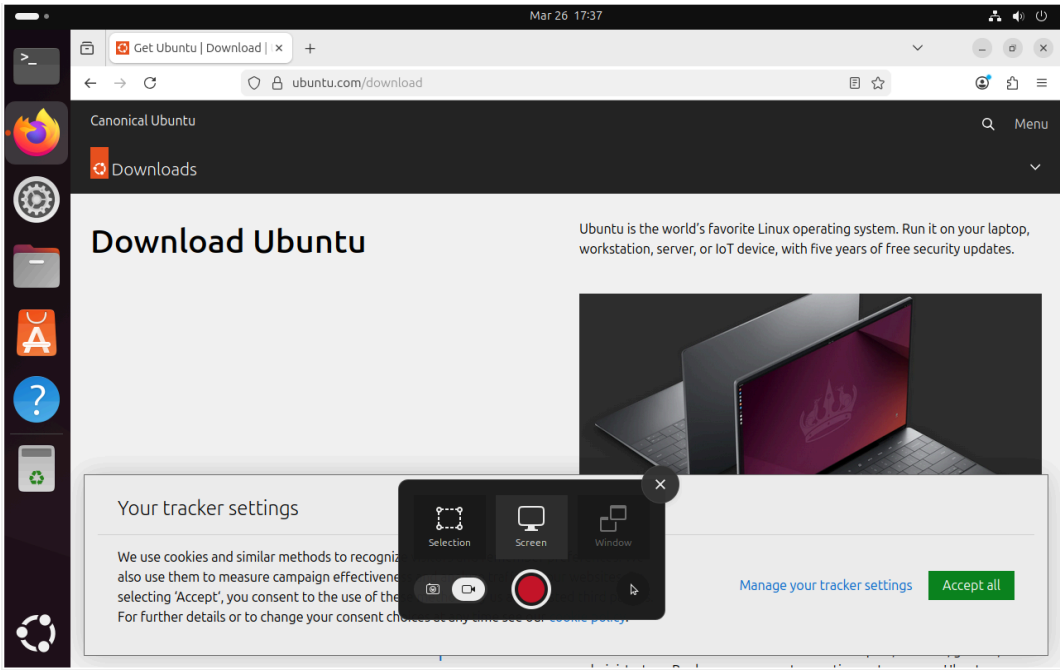


Figure. Ready for Screen Recording

Organizing a Practical Desktop Workflow

Good file management is not just about individual commands or menus. It is about building a workflow that reduces friction every day. The Ubuntu Academy outline frames this as **workflow optimization**, and the attached references support that approach by emphasizing launcher efficiency, workspaces, search, bookmarks, desktop features, and practical use of Linux as an everyday desktop system.

A practical Ubuntu workspace usually benefits from a few simple habits:

- keep **Downloads** temporary, not permanent
- move project files into clearly named folders quickly
- use **Bookmarks** for active work locations
- archive finished projects or move them into **Exports**, **Archive**, or **Completed** folders
- keep the **Desktop** visually clean
- use **Search** rather than wandering through deep folder trees
- keep screenshots and shared files in dedicated folders
- eject removable devices safely
- use Trash as a safety layer, not as a storage system

A good illustration is to imagine your desktop workflow as a kitchen. If ingredients, tools, leftovers, and serving plates are all mixed together, cooking becomes frustrating. If each item has a place, work becomes smoother and faster.

CLI example: create a simple daily workspace structure

```
mkdir -p ~/Workspace/{Active,Archive,Share,Screenshots}
tree -d ~/Workspace
```

Example output:

```
/home/student/Workspace
├── Active
├── Archive
├── Screenshots
└── Share
```

CLI example: move finished work into an archive folder

```
mv ~/Documents/Ubuntu-Academy ~/Workspace/Archive/
ls ~/Workspace/Archive
```

Example output:

```
Ubuntu-Academy
```

CLI example: search the workspace for PDFs only

```
find ~/Workspace -name "*.pdf"
```

Example output:

```
/home/student/Workspace/Archive/Ubuntu-Academy/Exports/final-report.pdf
```

This kind of workflow design is exactly what turns Ubuntu Desktop into a reliable daily platform instead of just a collection of applications.

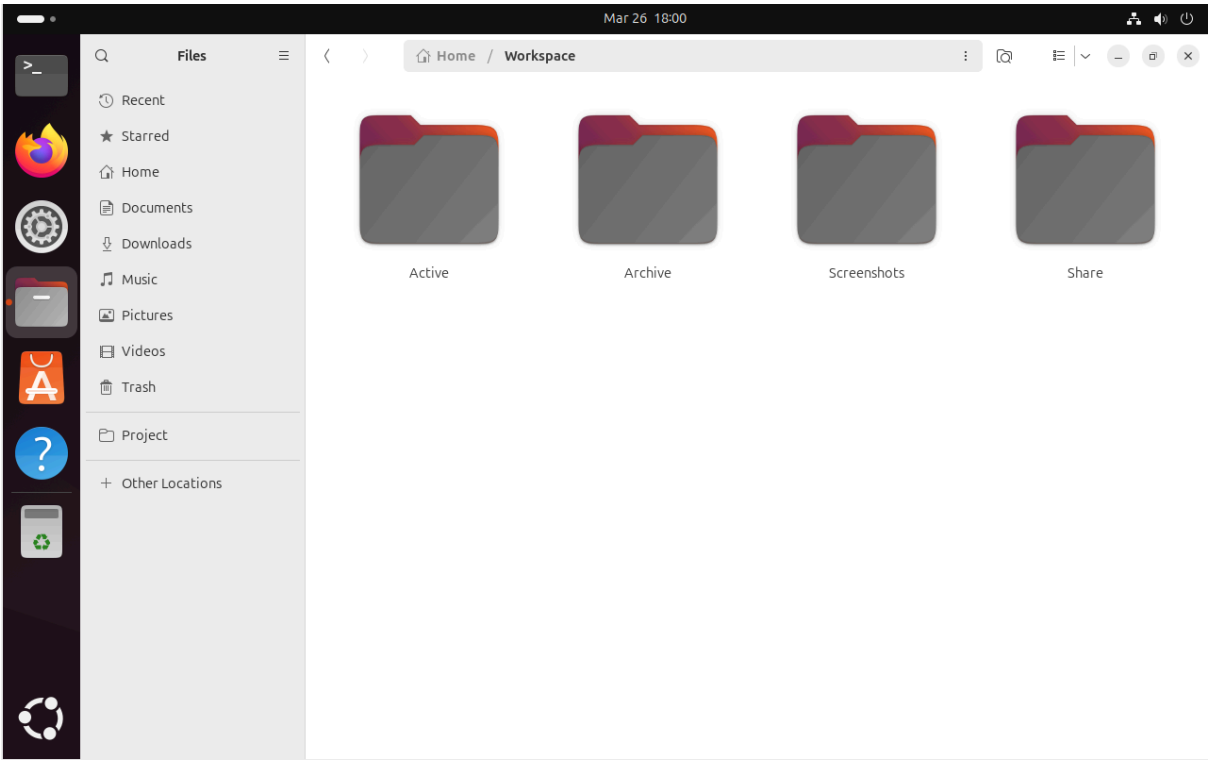


Figure. Files window showing a well-structured project hierarchy: Active, Archive, Share, Screenshots.

Guided File Management Activity

To make this chapter practical, readers can complete a short workspace lab:

- open **Files** and explore the Home directory
- create a project folder with subfolders for **Drafts**, **Images**, **Exports**, and **References**
- bookmark that folder in the sidebar
- search for a document using both **Files search** and `find`
- connect a USB drive and copy one project file to it
- create a ZIP archive of a folder
- move a file to **Trash** and restore it
- take a screenshot and save it into a dedicated Screenshots folder
- reorganize Downloads so that recent files are sorted into proper folders

This kind of exercise helps readers move from basic navigation to **intentional workspace management**.

Chapter Reflection

Files, search, and daily workspace management are among the most important desktop skills on Ubuntu. The Files application gives users a visual gateway into the Linux filesystem, while bookmarks, tabs, and structured folders make navigation faster and more intentional. Search tools in both the GUI and command line help users find information quickly, whether through indexed search with `plocate` or live filesystem searches with `find`. Archives make sharing and storage more efficient. External drives extend the workspace but require careful handling and safe ejection. Trash provides a recovery buffer before permanent deletion. Screenshots and recordings support communication, teaching, and troubleshooting. Together, these habits turn Ubuntu Desktop into a clean, dependable, and productive working environment.

Key Takeaways

- **Files (Nautilus) is the main visual bridge to the Linux filesystem on Ubuntu Desktop.**
- **Bookmarks, tabs, and folder structure** greatly improve everyday efficiency.
- **`plocate` and `find`** are powerful complements to graphical search.
- **Archives and compression** make sharing and storage more practical.
- **External drives should be mounted, used, and ejected carefully.**
- **Trash is a safety net**, while `rm` is permanent and should be used carefully.
- **Screenshots and recordings** are part of modern desktop communication and troubleshooting.
- **A clean workflow is built through habits**, not only through tools.

Part V — Users, Permissions, and Local Administration

Chapter 11 - Managing Users

A desktop computer often feels personal, but Ubuntu Desktop is designed to support **more than one person, more than one role, and more than one level of trust**. That is why user accounts and permissions matter. In the Ubuntu Academy outline, **creating and managing users, groups and sudo, password policies, GUI and CLI permissions, shared folders, and guest-style usage** are core skills for Ubuntu Desktop administration.

The attached references reinforce this same direction from different angles. They explain that Ubuntu was built to be easier for desktop users through graphical tools while still keeping Linux's underlying permission model intact. They also show that regular users are intentionally limited, that administrative work is normally done through **sudo**, and that file sharing becomes more powerful when users understand **ownership, groups, ACLs, and collaborative directories**.

A useful illustration is to imagine Ubuntu Desktop as a shared building. Each person has their own room, some rooms are shared, and a few doors require a master key. User accounts define **who each person is**, permissions define **which doors they can open**, and sudo is the supervised key that temporarily grants access to restricted areas. Once readers understand that model, desktop administration becomes much easier and much safer.

Creating and Managing Users

Ubuntu Desktop supports both graphical and command-line user management. The Ubuntu Academy outline explicitly calls out **Settings GUI**, `adduser`, `usermod`, `passwd`, and `deluser` as the practical tools readers should know.

This dual approach is one of Ubuntu's strengths. Beginners may start from **Settings > Users**, where accounts can be added, removed, or adjusted visually. More advanced users or classroom administrators may prefer the command line because it is faster, more reproducible, and easier to document.

Creating separate user accounts is important because it gives each person:

- a personal **Home folder**
- their own desktop preferences
- private browser data, files, and settings
- controlled access to software and administrative tasks

The references also point out an important privacy improvement in Ubuntu: users' Home folders are treated as private by default, which means one user should not

casually browse another user's personal files unless permissions are intentionally changed.

A practical desktop rule is simple: **one person, one account**. Shared passwords and shared accounts may seem convenient, but they quickly create confusion, privacy issues, and security risks.

CLI example: create a new user

```
sudo adduser student2
```

Example output:

```
Adding user `student2' ...
Adding new group `student2' (1002) ...
Adding new user `student2' (1002) with group `student2'
...
Creating home directory `/home/student2' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for student2
Enter the new value, or press ENTER for the default
        Full Name []: Student Two
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
```

This is a friendly command for creating users because it guides the admin through password and profile prompts.

CLI example: list the user's home directory

```
ls /home
```

Example output:

```
student  student2
```

This reminds readers that each account normally gets its own workspace under `/home`.

CLI example: modify a user account

```
sudo usermod -aG sambashare student2
```

Typical result:

```
(no output if successful)
```

This adds the user to another group without removing existing memberships.

CLI example: remove a user account

```
sudo deluser student2
```

Example output:

```
Removing user `student2' ...
Done.
```

These CLI tools directly match the Ubuntu Academy outline and help readers understand that desktop user management is not limited to the GUI.

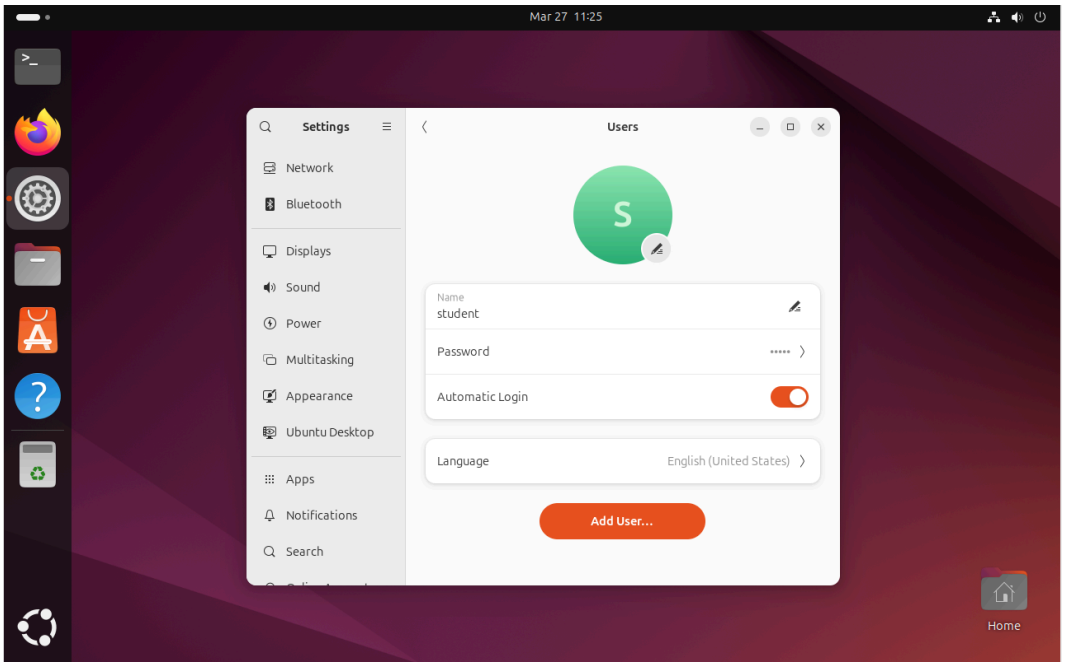


Figure. Settings > Users showing the current account and an option to add another user.

Password Changes and Account Recovery Basics

Passwords are the first line of protection for a desktop account. In the Ubuntu Academy outline, user security includes **password policies**, account protection, and practical desktop safety.

On Ubuntu Desktop, password changes may happen through the graphical Users settings or through the `passwd` command. This matters because users sometimes forget passwords, need to rotate them, or must reset credentials in a classroom or shared-computer environment.

A good password policy for desktop use should encourage:

- passwords that are **hard to guess**
- avoiding reused passwords across systems
- changing a password if compromise is suspected
- not sharing account passwords casually, even with trusted colleagues

A helpful illustration is to think of the password as the key to a private office. A strong door is useful, but if the key is copied everywhere, privacy disappears.

CLI example: change the current user's password

```
passwd
```

Example output:

```
Changing password for student.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

CLI example: administrator changes another user's password

```
sudo passwd student2
```

Example output:

```
New password:
Retype new password:
passwd: password updated successfully
```

This is useful in labs, schools, or support scenarios.

CLI example: check who you are before resetting anything

```
whoami
```

Example output:

```
student
```

The Mastering Ubuntu reference also uses `whoami` in its sudo examples, which makes it a natural teaching companion here.

For account recovery, desktop users should start with the simplest path:

- verify the correct username
- confirm keyboard layout and Caps Lock state
- use the admin account to reset the password if available
- preserve user privacy when helping others regain access

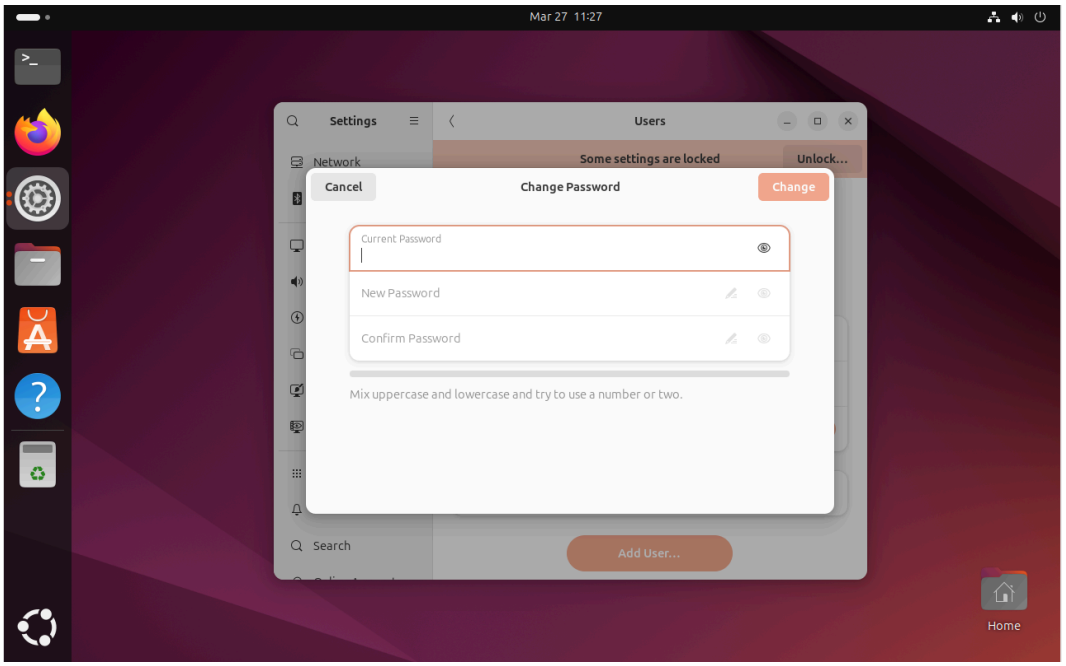


Figure. Password change dialog in Settings > Users.

User Switching and Multi-User Desktops

Ubuntu Desktop supports multiple local users cleanly, which makes it suitable for homes, offices, labs, and shared learning environments. The Ubuntu Academy outline explicitly includes **user switching, multi-user desktops, and guest-style workflows**.

Multi-user desktop use means Ubuntu must handle:

- separate sessions
- separate files and browser data
- fast switching between users
- session locking and privacy between accounts

The references also emphasize that Ubuntu's desktop has a user/session control area and that user-specific Home folders are private by design. That is especially important on shared machines: switching users should not mean exposing someone else's work.

A practical way to explain this is to imagine several people using the same library study room at different times. Each person can have their own locked drawer, even though the room itself is shared.

CLI example: show the current logged-in user

```
whoami
```

Example output:

```
student
```

CLI example: list currently logged-in sessions

```
who
```

Example output:

```
student  tty2         2026-03-14 09:05
student2 tty3         2026-03-14 09:12
```

CLI example: list session information through loginctl

```
loginctl list-sessions
```

Example output:

```
SESSION  UID USER     SEAT  TTY
     2 1000 student  seat0 tty2
     3 1002 student2 seat0 tty3
```

These commands help readers see that multi-user desktops are not only a visual feature but a real session-management system underneath.

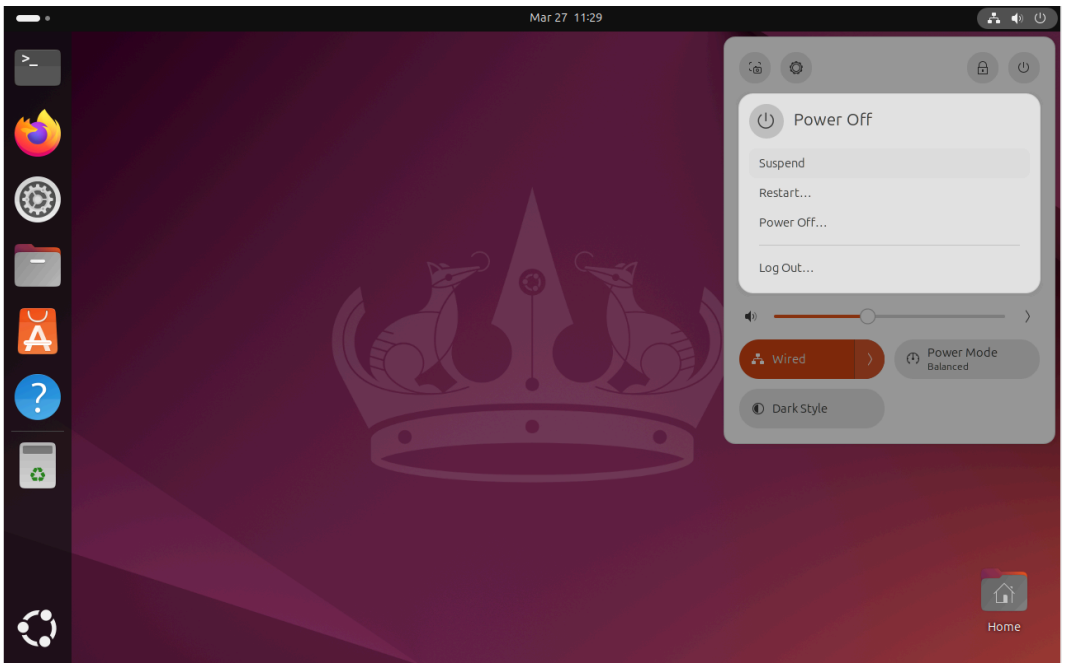


Figure. System menu showing Switch User, Lock, and Log Out options.

Groups and Administrative Privileges

Groups are one of the main ways Linux organizes permission and responsibility. The Ubuntu Academy outline specifically highlights **groups and sudo**, and the attached references go further by showing that users may need to be added to groups such as `sudo`, `adm`, `lpadmin`, or `sambashare` depending on what they need to manage.

A group is like a team badge. Instead of assigning every permission individually to every person, Ubuntu can assign a capability to a group and then place the appropriate users into that group.

Common desktop-relevant groups may include:

- `sudo` for administrative actions
- `lpadmin` for printer management
- `sambashare` for certain sharing-related tasks
- `adm` for access to some system logs or admin contexts

The Mohr reference provides a very practical example: if a user is not already in the `sudo` group, an admin can add them with:

```
sudo adduser <username> sudo
```

and can add them to multiple groups such as `adm sudo lpadmin sambashare`.

CLI example: show a user's groups

```
groups
```

Example output:

```
student adm cdrom sudo dip plugdev lpadmin sambashare
```

CLI example: add a user to the sudo group

```
sudo adduser student2 sudo
```

Example output:

```
Adding user `student2' to group `sudo' ...
Adding user student2 to group sudo
Done.
```

CLI example: add a user to several desktop-related groups

```
sudo adduser student2 adm sudo lpadmin sambashare
```

Example output:

```
Adding user `student2' to group `adm' ...
Adding user `student2' to group `sudo' ...
Adding user `student2' to group `lpadmin' ...
Adding user `student2' to group `sambashare' ...
Done.
```

This example follows the exact group-management pattern described in the Mohr reference.

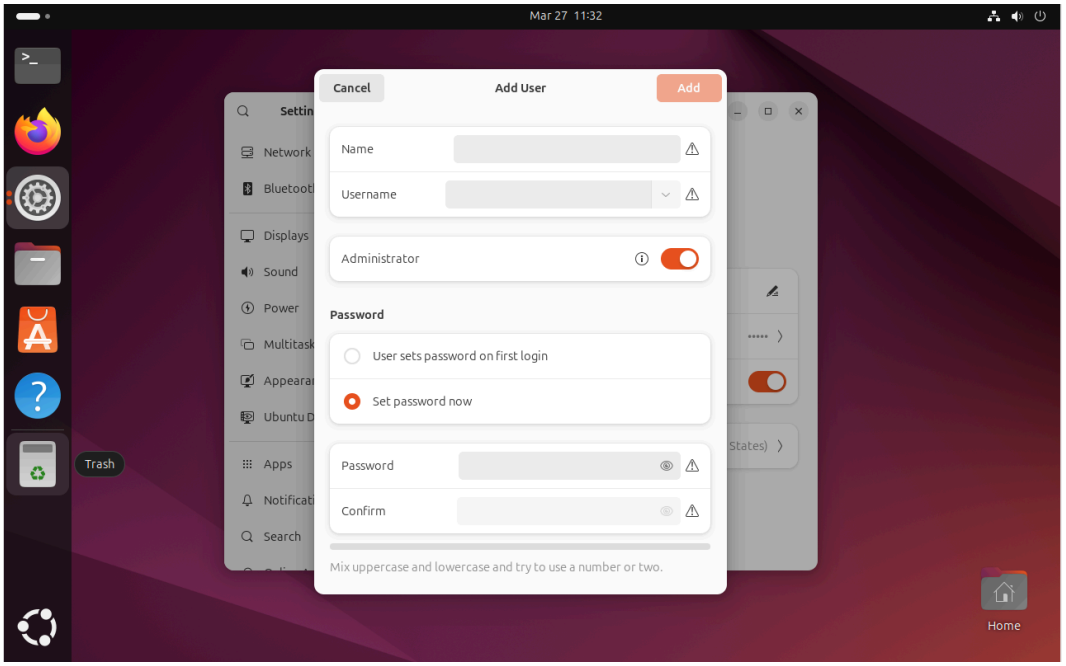


Figure. Settings > Users showing Standard vs Administrator account type.

sudo Basics in Desktop Administration

The references are very clear that Ubuntu users should normally use **sudo** instead of signing in directly as root. The Mohr reference explains that Ubuntu's default user is an administrative user who can precede restricted commands with `sudo`, and it specifically recommends using `sudo` rather than a full root shell because the root account is locked in Ubuntu for security reasons. The Mastering Ubuntu reference adds that `sudo` grants temporary administrative privileges and records a timestamp, allowing repeated use for a short period without asking for the password every single time.

This is an important desktop safety principle: **do normal work as a normal user, and elevate only when needed**.

Typical desktop admin tasks that require sudo include:

- installing software
- changing system-wide settings
- adding users
- modifying protected files
- managing services or hardware settings

A good illustration is to think of sudo as borrowing the master key from a supervisor desk for one specific task, then returning to normal access as soon as possible.

CLI example: update the system with sudo

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Reading package lists... Done
```

CLI example: what happens without sudo on a restricted action

```
apt-get update
```

Example output:

```
Reading package lists... Done
E: Could not open lock file /var/lib/apt/lists/lock -
open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
```

This reflects the same style of permission-denied example shown in Mastering Ubuntu, where running APT without root privileges fails.

CLI example: validate or refresh sudo credentials

```
sudo -v
```

Typical result:

```
(no output if successful)
```

The Mastering Ubuntu reference explicitly documents `sudo -v` as refreshing the time limit without running a command.

CLI example: list a user's sudo privileges

```
sudo -l
```

Example output:

```
Matching Defaults entries for student on ubuntu-desktop:
    env_reset, mail_badpass, secure_path=...

User student may run the following commands on
ubuntu-desktop:
    (ALL : ALL) ALL
```

This is a valuable administrative check when troubleshooting privilege issues.

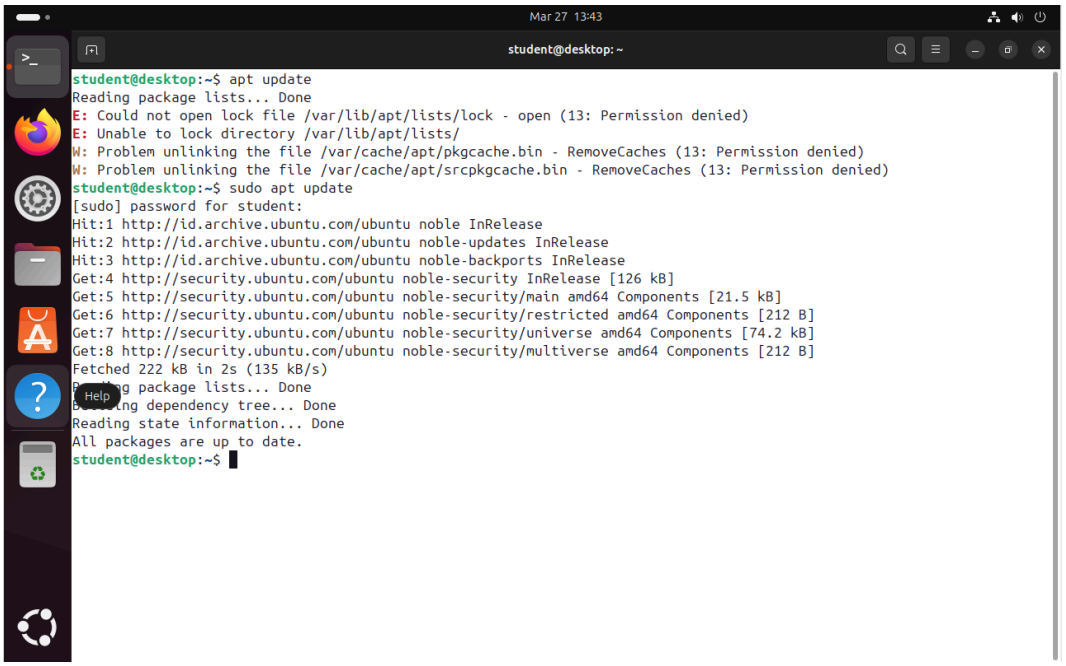


Figure.Terminal showing sudo apt update succeeding after a non-sudo attempt failed.

File and Folder Permissions for Desktop Users

Linux file permissions are one of the most important concepts for desktop administration. The Ubuntu Academy outline explicitly includes **GUI properties**, `chmod`, and `chown` in desktop context.

At the simplest level, each file and folder has three main permission categories:

- **user** — the owner
- **group** — the assigned group
- **others** — everyone else

And each category can have combinations of:

- **read** (`r`)
- **write** (`w`)
- **execute** (`x`)

The Mohr reference reminds readers that `ls -l` shows detailed information including permissions and ownership, which makes it one of the best first tools for learning how the permission model works.

A helpful illustration is to imagine a filing cabinet with three labels: owner, team, and visitors. Each label has its own rules about who may open, edit, or use the contents.

CLI example: inspect file permissions

```
ls -l ~/Documents
```

Example output:

```
-rw-rw-r-- 1 student student  4210 Mar 14 09:20
report.txt
drwxrwxr-x 2 student student  4096 Mar 14 09:25 ProjectA
```

This shows both a file and a directory, along with owner, group, and permission bits.

CLI example: make a script executable

```
chmod +x ~/Documents/runme.sh
ls -l ~/Documents/runme.sh
```

Example output:

```
-rwxrwxr-x 1 student student 128 Mar 14 09:31
/home/student/Documents/runme.sh
```

CLI example: change file ownership

```
sudo chown student2:student2 ~/Documents/shared-note.txt
ls -l ~/Documents/shared-note.txt
```

Example output:

```
-rw-rw-r-- 1 student2 student2 204 Mar 14 09:34
/home/student/Documents/shared-note.txt
```

A beginner-friendly lesson here is that **ownership decides the default authority**, while permissions decide the level of access.

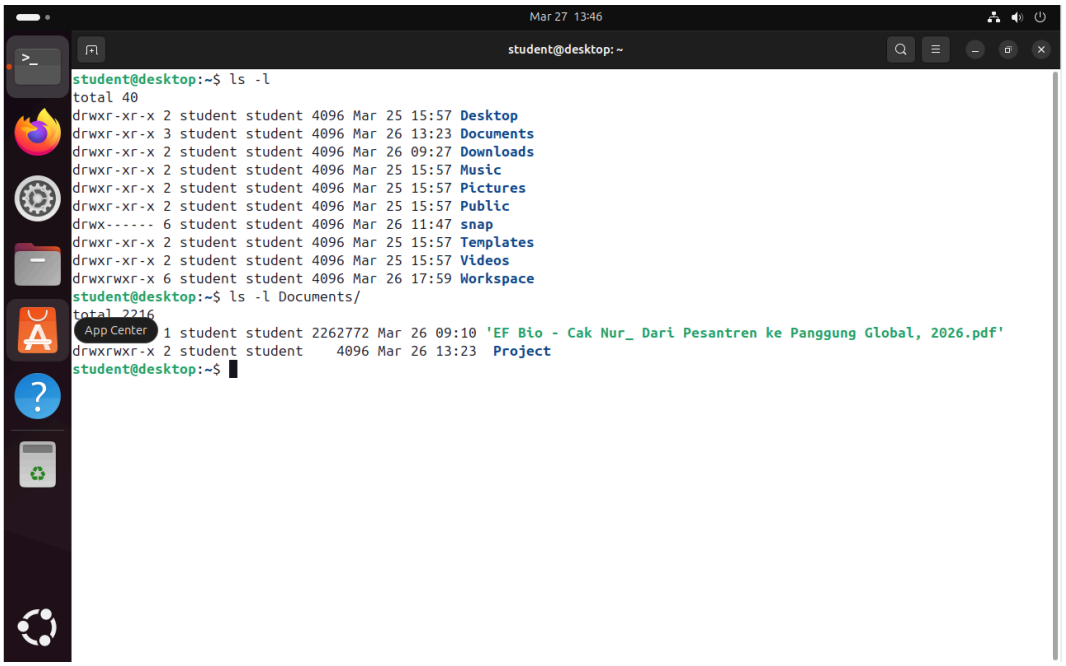


Figure. Terminal showing ls -l output for a document folder.

Ownership, Sharing, and Collaborative Folders

Sharing files on a desktop is easy when only one user is involved. It becomes more interesting when multiple users need controlled access. The Ubuntu Linux Bible explains that Linux's classic permission model is not always flexible enough for modern collaboration, and it points directly to **shared directories**, **setgid directories**, **sticky bit behavior**, and especially **ACLs** as tools for selective sharing. It also explains that ACLs allow owners to grant read, write, and execute rights to specific users or groups without opening the file to everyone or changing the main ownership model.

The Mohr reference complements this from the desktop side by showing practical **Samba/SMB folder sharing** through Files and network locations, including bookmarking network folders and handling username/password prompts where necessary.

This gives Ubuntu Desktop users two major sharing patterns:

- **local collaboration** between users on the same Ubuntu system
- **network sharing** across machines using SMB/Samba and similar workflows

A practical illustration is to imagine a team whiteboard room. Some documents belong only to one person, some belong to the group, and some are shared through a common network shelf that everyone in the team can reach.

CLI example: create a shared project folder

```
sudo mkdir -p /srv/shared-project
sudo chown student:student /srv/shared-project
ls -ld /srv/shared-project
```

Example output:

```
drwxr-xr-x 2 student student 4096 Mar 14 09:40
/srv/shared-project
```

CLI example: grant another user ACL permissions

```
setfacl -m u:student2:rwx /srv/shared-project
getfacl /srv/shared-project
```

Example output:

```
# file: srv/shared-project
# owner: student
# group: student
user::rwx
user:student2:rwx
group::r-x
mask::rwx
other::r-x
```

This directly follows the Ubuntu Linux Bible's explanation of `setfacl` and `getfacl` for selective file sharing.

CLI example: access a Samba share from the desktop side

```
xdg-open smb://192.168.1.10/shared
```

Typical result:

```
(opens the Files application at the remote SMB location)
```

The Mohr reference specifically describes using **Files > Other Locations** and even typing an `smb://` address to reach a remote shared folder.

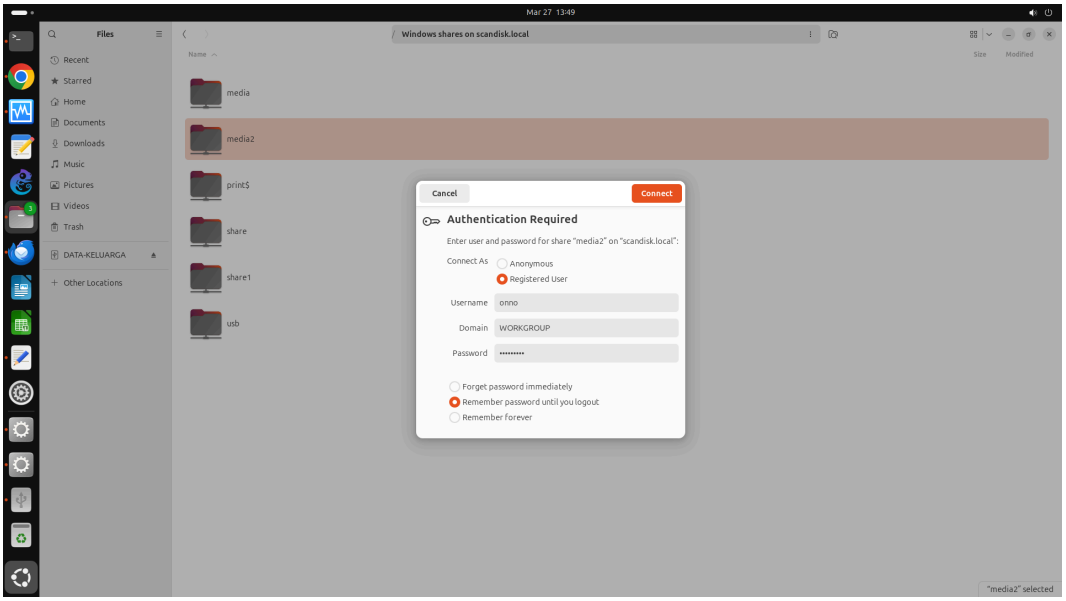


Figure. Files > Other Locations showing an SMB network share.

Guest-style Usage, Privacy, and Session Hygiene

Even when Ubuntu does not always provide a traditional guest account in every desktop edition, the Ubuntu Academy outline rightly emphasizes **guest sessions, privacy, and session hygiene** as user-management topics.

The underlying principle is more important than the label: sometimes a machine needs **temporary, low-risk use**. That might involve a shared lab computer, a family machine, or a classroom demonstration system. The goal is to let someone use the desktop without exposing another user's data or leaving the system in a messy state.

Good session hygiene includes:

- locking the screen when stepping away
- logging out after use on shared systems
- not storing private files in another user's account
- clearing temporary downloads or browser sessions on shared machines
- using a standard user account rather than an admin account for routine work
- respecting the privacy of other Home folders, which Ubuntu increasingly protects by default

A helpful illustration is to think of session hygiene like cleaning a study desk before leaving the library. Good habits protect both the user and the next person.

CLI example: lock the current session

```
loginctl lock-session
```

Typical result:

```
(no output if successful)
```

CLI example: check the current desktop session identity

```
echo $XDG_CURRENT_DESKTOP
echo $DESKTOP_SESSION
```

Example output:

```
ubuntu:GNOME
ubuntu
```

CLI example: see what autostarts or personal files might remain visible

```
ls ~/Downloads
ls ~/Desktop
```

Example output:

```
meeting-link.txt  screenshot.png
project-shortcut.desktop
```

This simple check teaches a good privacy lesson: what is left behind in visible folders often tells the story of whether a session was handled cleanly.

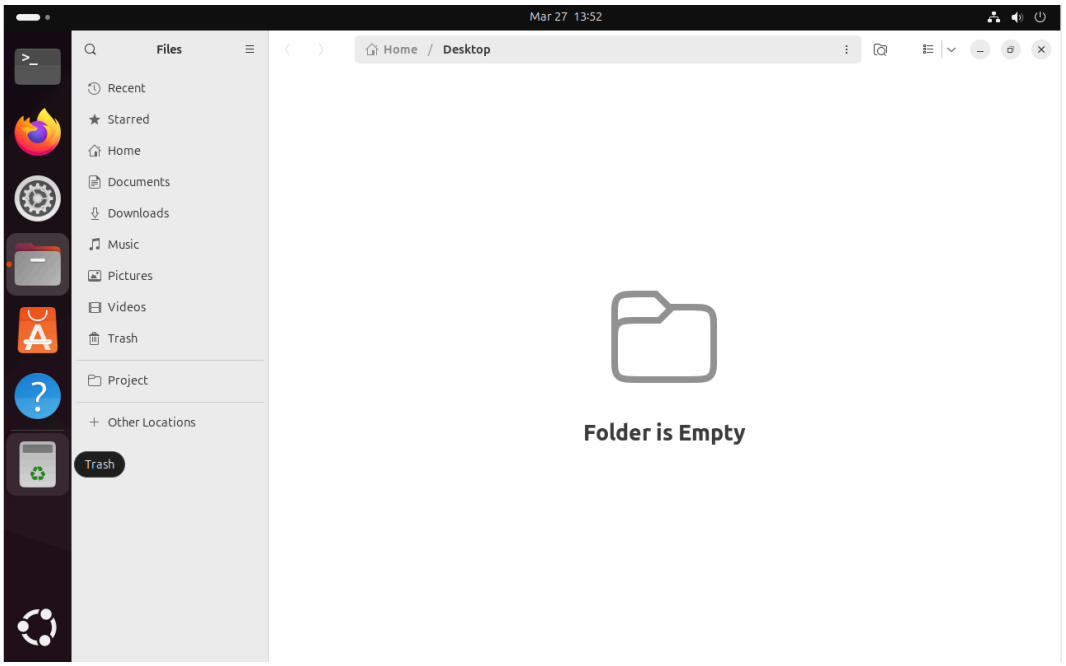


Figure. Show Desktop in Files

Guided User and Permissions Lab

To make this chapter practical, readers can complete a short hands-on exercise:

- create a second user account with `sudo adduser`
- verify that each user has a separate Home directory
- switch users from the desktop session menu
- run `groups` and identify whether the current account belongs to `sudo`
- add the second user to `lpadmin` or `sambashare`
- create a shared folder and inspect its ownership with `ls -ld`
- grant another user access using `setfacl`
- change a file's owner with `chown`
- change a file's permissions with `chmod`
- lock the session and sign back in

This kind of exercise turns abstract permissions into visible, repeatable desktop skills.

Chapter Reflection

User accounts and permissions are the foundation of safe and practical Ubuntu Desktop administration. Ubuntu supports multiple users through both graphical tools and command-line commands such as `adduser`, `usermod`, `passwd`, and `deluser`. Password changes and recovery basics help keep accounts usable and secure. Multi-user desktops rely on user switching, separate Home folders, and session privacy. Groups organize capabilities efficiently, while `sudo` provides temporary administrative access without requiring daily work as root. File permissions, ownership, and tools such as `chmod`, `chown`, `setfacl`, and `getfacl` make it possible to control access carefully, whether for single-user privacy or collaborative sharing. Finally, good session hygiene—locking, logging out, and cleaning visible data—helps Ubuntu remain professional and trustworthy on shared systems.

Key Takeaways

- **Ubuntu Desktop supports both GUI and CLI user management.**
- **Each user should normally have their own account and Home directory.**
- **Groups simplify permission management**, especially for admin, printing, and sharing tasks.
- **`sudo` is the normal and recommended way** to perform administrative actions on Ubuntu.
- **`ls -l`, `chmod`, and `chown`** are essential tools for understanding and managing permissions.
- **ACLs provide selective sharing** beyond the traditional Linux owner/group/other model.
- **Good privacy and session hygiene matter**, especially on shared desktops.

Chapter 12 — Desktop Administration Tools

A good Ubuntu Desktop user can open applications, manage files, and customize the interface. A good Ubuntu Desktop administrator goes one step further: they know **which tool to use, when to use it, and how to solve everyday problems safely**. In the Ubuntu Academy outline, this chapter sits naturally between user management and later hardware/networking/security topics because it focuses on the practical tools that help keep a desktop healthy, understandable, and supportable. The outline explicitly emphasizes **GUI vs command-line workflows, startup/session management, troubleshooting frozen apps, slow boots, performance tuning, and everyday best practices**.

The attached Ubuntu references reinforce this direction strongly. They show Ubuntu as a desktop that includes **core graphical tools** such as the Settings app, file manager, terminal emulator, tweak tools, and System Monitor, while also making it clear that the **shell** remains the more powerful professional interface for many advanced tasks. They also describe tools such as `ps`, `top`, and **System Monitor** for handling processes, and they explain how startup applications, multi-user sessions, and desktop support can be managed in practical ways.

A helpful illustration is to imagine Ubuntu Desktop administration as maintaining a classroom or lab. Some tasks are best done with clearly labeled wall switches and control panels. Others require opening the maintenance cabinet and using specialized tools. Ubuntu gives administrators both kinds of tools. The real skill is knowing how to move between them with confidence and restraint.

Settings App vs Command-Line Administration

One of Ubuntu Desktop's greatest strengths is that it does **not** force users into a single administrative style. The system provides a rich graphical **Settings** application and related desktop tools, but it also preserves the Linux command-line tradition for tasks that require precision, automation, or deeper visibility. The Ubuntu Academy outline explicitly calls this out through topics such as **Software Updater vs command line**, **Settings GUI**, **startup applications**, **hardware queries**, and **performance tuning**.

The attached references explain the same pattern in practical terms. Ubuntu The Complete Guide describes **System settings** as the place for changing keyboard, mouse, time, date, timezone, and other desktop behavior, including shortcuts and startup applications. At the same time, Ubuntu Linux Bible emphasizes that once users move into more professional administration, the **shell** becomes essential because many advanced capabilities are simply easier and more powerful there.

A practical rule for desktop administration is:

- use the **GUI** when the task is common, clear, and visual
- use the **Terminal** when the task needs precision, repeatability, or deeper inspection

Typical GUI-friendly tasks include:

- changing display, power, and keyboard settings
- connecting to Wi-Fi
- adjusting appearance
- switching users
- checking startup apps
- reading a graphical process list in System Monitor

Typical Terminal-friendly tasks include:

- checking detailed process and system status
- running scripted or repeatable admin actions
- changing permissions and ownership
- querying hardware and services
- troubleshooting package or log issues
- documenting exact steps for labs or support notes

A useful illustration is this: the Settings app is like the dashboard of a vehicle. The Terminal is like opening the service panel under the hood. Both matter, but they serve different depths of control.

CLI example: identify the desktop session before troubleshooting

```
echo $XDG_CURRENT_DESKTOP
echo $DESKTOP_SESSION
```

Example output:

```
ubuntu:GNOME
ubuntu
```

CLI example: open the graphical Settings app from Terminal

```
gnome-control-center &
```

Example output:

```
[1] 5028
```

This is a simple but useful bridge between GUI and CLI administration.

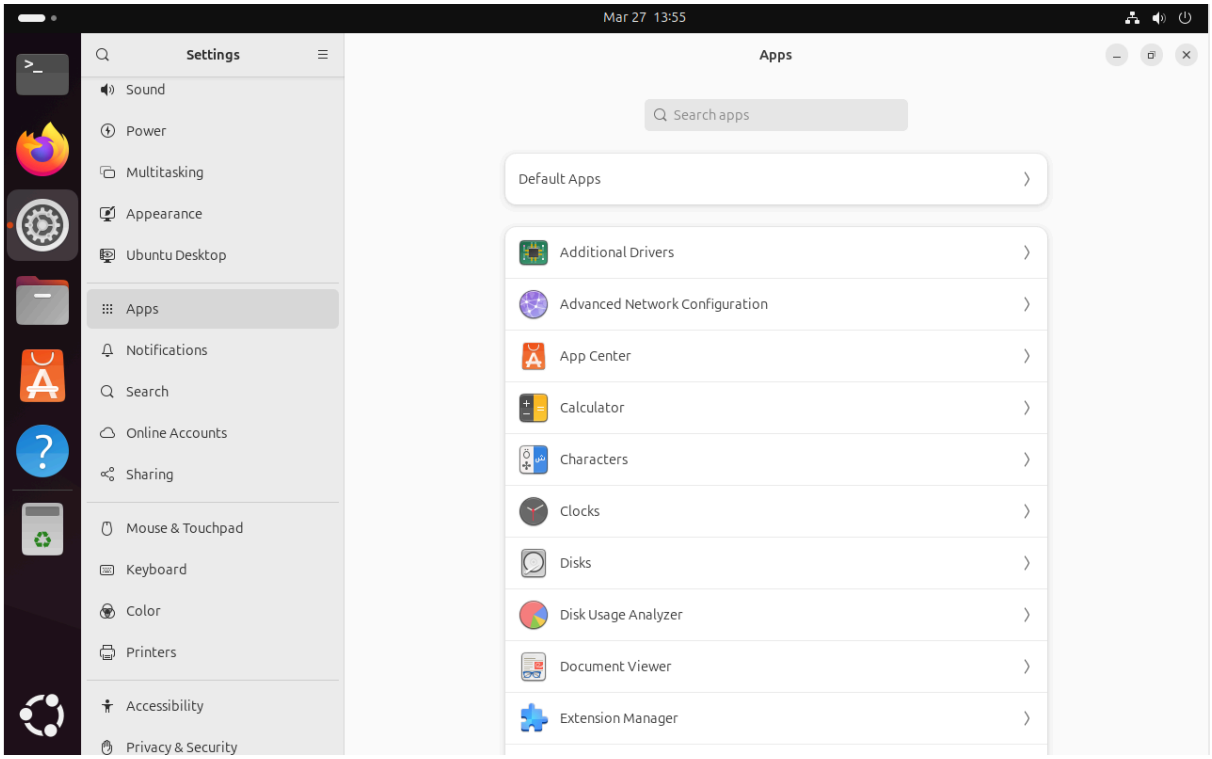


Figure. Ubuntu Settings app open on the main overview page.

Privilege Escalation Safely

Desktop administrators often need to perform tasks that ordinary users should not perform by default. Ubuntu's standard answer to this is **sudo**, not permanent root login. The attached references explain this clearly: Ubuntu expects users to work normally as a regular user and temporarily elevate for administrative tasks. The Mohr guide notes that Ubuntu's default user is an administrative user who can prepend restricted commands with `sudo`, while Mastering Ubuntu emphasizes that `sudo` grants temporary elevated privileges and maintains a short authentication timestamp for convenience.

This matters because safe administration is not just about knowing commands. It is about knowing **how much power to use and for how long**. Running everything with administrative privileges all the time increases risk, especially on desktop systems where web browsing, file downloads, and experimentation are common.

A good privilege-escalation mindset is:

- do everyday work as a **regular user**
- use `sudo` only for specific administrative actions
- read commands carefully before pressing Enter
- avoid random copy-paste from untrusted sources
- prefer one targeted admin command over opening a full root shell unnecessarily

A helpful illustration is to think of sudo as borrowing the headmaster's key for one room, not wearing the full master-key ring around your neck all day.

CLI example: run an administrative command properly

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Reading package lists... Done
```

CLI example: see what happens without sudo

```
apt update
```

Example output:

```
Reading package lists... Done
E: Could not open lock file /var/lib/apt/lists/lock - open
(13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
```

This is the same basic administrative lesson shown in the Ubuntu references: restricted system actions need elevated privileges.

CLI example: refresh sudo credentials without running another command

```
sudo -v
```

Typical result:

```
(no output if successful)
```

Mastering Ubuntu explicitly documents `sudo -v` as a way to validate or refresh the current sudo timestamp.

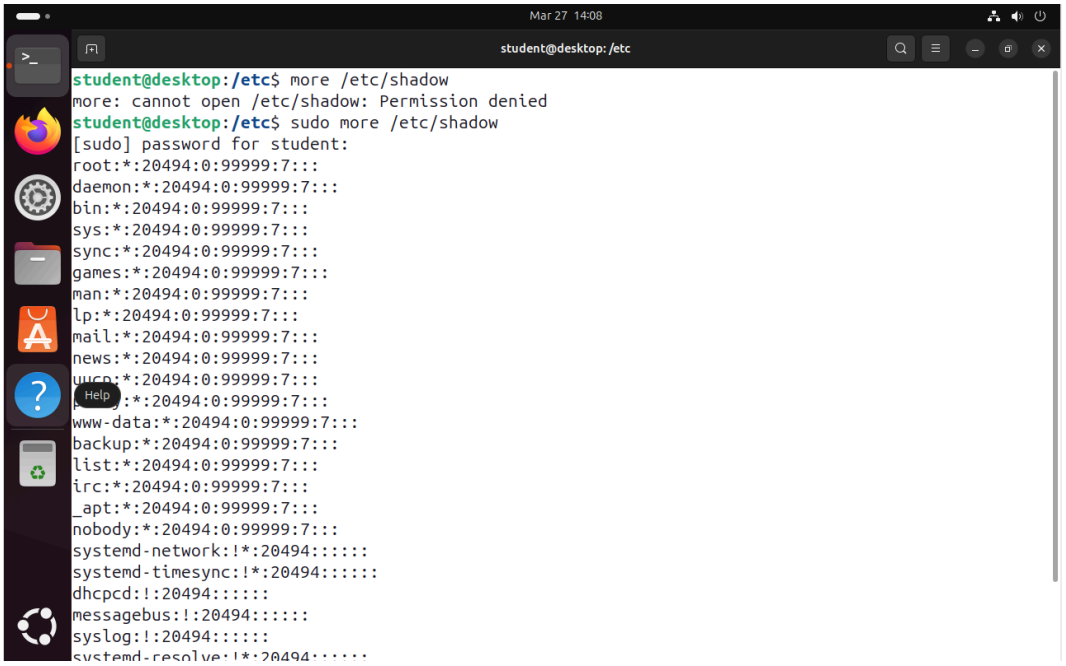


Figure. Terminal showing a permission-denied result without sudo and success with sudo.

Logs, Status, and Basic Diagnostic Commands

When something goes wrong on Ubuntu Desktop, the most useful question is often not “What do I click?” but **“What does the system say?”** Good diagnostics begin with logs, status output, and a few trustworthy commands. The Ubuntu Academy outline explicitly mentions **frozen apps, slow boot, troubleshooting connectivity, and common issues**, which all depend on basic diagnostic literacy.

Although the attached references focus most heavily on process tools and desktop utilities, they also reinforce the general idea that the Linux shell gives access to professional troubleshooting methods beyond what the graphical desktop shows. The shell chapter in Ubuntu Linux Bible emphasizes that many advanced features are only available or more accessible through shell commands, which makes it the natural place for status and log inspection.

A strong beginner diagnostic toolkit includes:

- `whoami` to confirm identity
- `hostname` to confirm the machine
- `uptime` to see how long the system has been running and its load
- `df -h` to check disk space
- `free -h` to check memory usage
- `journalctl` to inspect system logs
- `dmesg` for kernel-related messages when appropriate

A good illustration is to think of diagnostics as reading the notes in a maintenance logbook before opening the machine. The system often already knows what happened; the admin’s job is to learn how to ask.

CLI example: check basic system identity and uptime

```
whoami
hostname
uptime
```

Example output:

```
student
ubuntu-desktop
 09:54:12 up 2:17,  2 users,  load average: 0.31, 0.42, 0.51
```

CLI example: inspect storage and memory quickly

```
df -h
free -h
```

Example output:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p2  236G   31G  194G  14% /

               total        used        free      shared  buff/cache   available
Mem:           7.6Gi       2.1Gi       3.8Gi       318Mi       1.7Gi       5.0Gi
Swap:          2.0Gi          0B       2.0Gi
```

CLI example: view recent system log entries

```
journalctl -n 20
```

Example output:

```
Mar 14 09:50:01 ubuntu-desktop gnome-shell[2412]: Window manager warning: ...
Mar 14 09:50:12 ubuntu-desktop systemd[1]: Started User Manager for UID 1000.
Mar 14 09:51:05 ubuntu-desktop NetworkManager[882]: <info>  [..] device connected
```

CLI example: show only current-boot logs

```
journalctl -b -n 15
```

Example output:

```
Mar 14 07:37:10 ubuntu-desktop kernel: Linux version ...
Mar 14 07:37:15 ubuntu-desktop systemd[1]: Started GNOME Display Manager.
Mar 14 07:37:30 ubuntu-desktop gdm-password][1014]:
pam_unix(gdm-password:session): session opened
```

These examples are especially helpful for desktop-support contexts where users report “it froze,” “it was slow,” or “the network failed.”

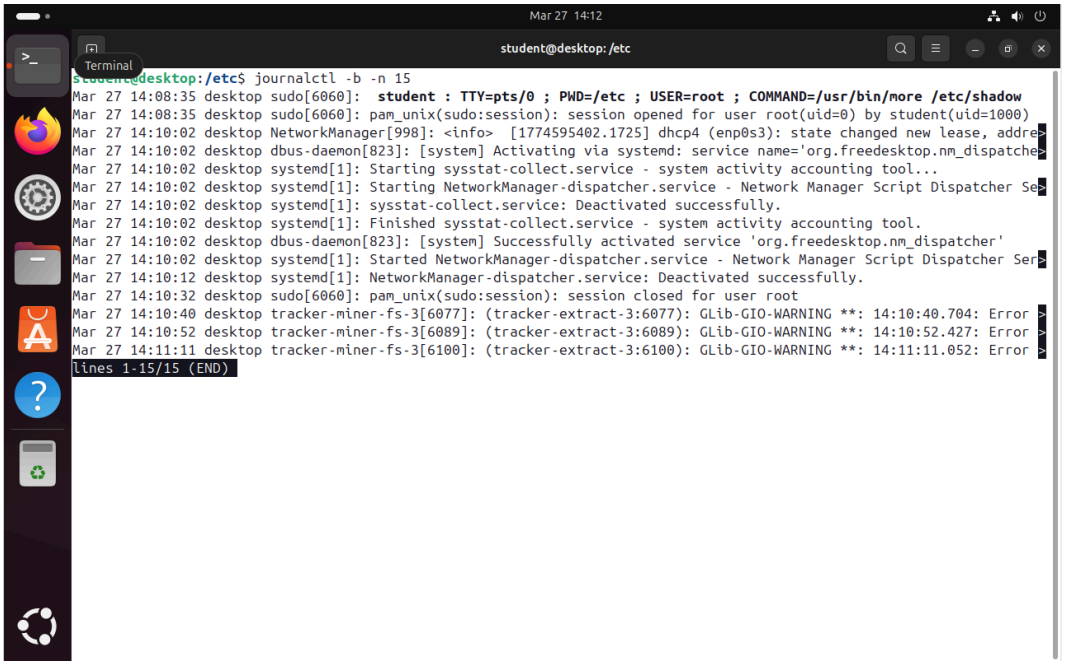


Figure. Terminal showing journalctl output for recent system events. Notice bold line.

Monitoring Processes and System Activity

The attached Ubuntu Linux Bible provides some of the strongest support for this section. It devotes a full chapter to **managing running processes** and explicitly describes `ps`, `top`, and the graphical **System Monitor** as the key tools for listing, sorting, understanding, and controlling processes on a Linux desktop. It also explains that processes can be killed or reniced, and that System Monitor gives GNOME users a graphical way to sort and manage them.

This is critical for desktop administration because many common support problems are really process problems:

- a browser tab consumes too much RAM
- a background app spikes CPU usage
- a frozen desktop app will not close
- a startup process slows login
- a misbehaving sync tool runs constantly

A practical monitoring progression is:

- start with **System Monitor** if you want a quick visual overview
- use `ps` for detailed listing and sorting
- use `top` for live monitoring
- kill or renice only after identifying the correct process carefully

A good illustration is to think of the desktop like a busy office. Processes are the people working inside it. Some are productive and quiet. Some are taking too much space or attention. Monitoring tools help you see who is doing what.

CLI example: list processes by memory size

```
ps -eo pid,user,group,gid,vsz,rss,comm --sort=-vsz |
head
```

Example output:

```
 PID USER     GROUP      GID    VSZ   RSS COMMAND
2366 student  student   1000 3720060 317060 gnome-shell
3030 student  student   1000 2456968 248340 firefox
3233 student  student   1000 2314388 316252 Web Content
```

This is drawn directly from the Ubuntu Linux Bible's process-management examples.

CLI example: live monitoring with top

```
top
```

Example output:

```
top - 10:02:11 up 2:25,  2 users,  load average: 0.48, 0.55, 0.52
Tasks: 286 total,   1 running, 285 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.4 us,  2.1 sy,  0.0 ni, 90.8 id,  0.3 wa,  0.0 hi,  0.4 si,  0.0 st
MiB Mem :   7800.0 total,   3890.0 free,   2110.0 used,   1800.0 buff/cache
```

The Ubuntu Linux Bible explains that `top` refreshes repeatedly, can sort, and can be used to kill or renice misbehaving processes.

CLI example: kill a stuck process

```
kill 3030
```

Typical result:

```
(no output if successful)
```

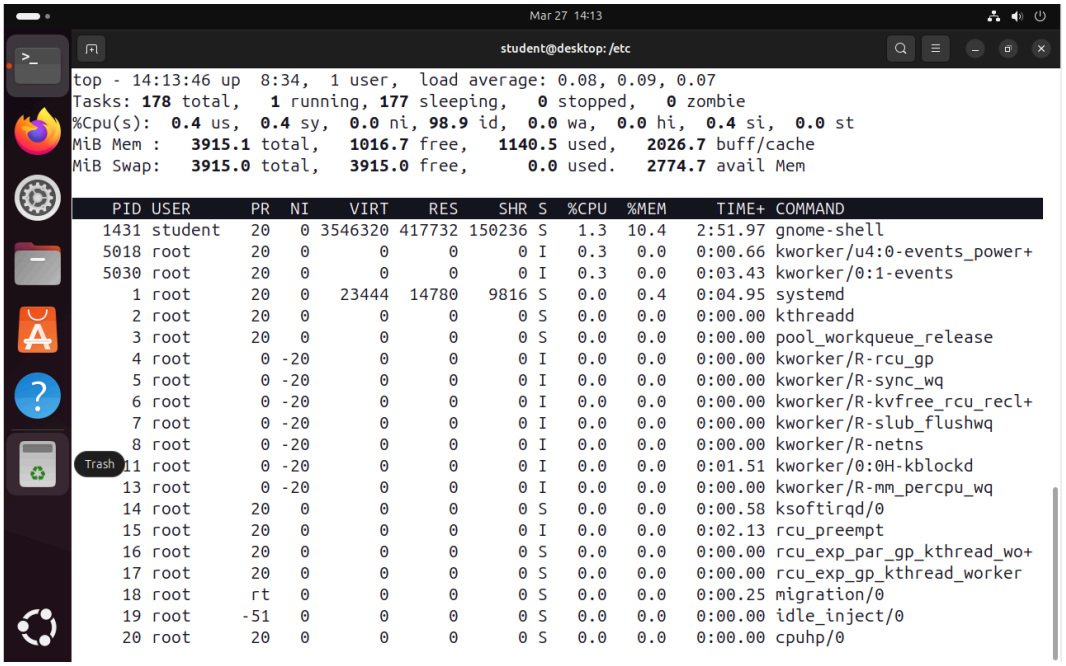
CLI example: graphical process monitor

```
gnome-system-monitor &
```

Example output:

```
[1] 6152
```

The Ubuntu Linux Bible specifically describes **System Monitor (gnome-system-monitor)** as GNOME's graphical process-management tool.



Figure, Terminal running top and showing CPU/memory activity.

Managing Startup Items and Sessions

Startup behavior strongly affects the desktop experience. A system that launches too many apps at login can feel slow, cluttered, or unstable. The attached Ubuntu Complete Guide is especially helpful here. It explicitly explains that Ubuntu can configure which applications start at login, that users can open **Startup Applications** from the Dash, and that they can also launch it with `gnome-session-properties`. It even provides a concrete example of adding Firefox to startup.

The Ubuntu Academy outline also directly includes **managing startup applications and sessions**, which makes this a core desktop-admin skill rather than a minor convenience feature.

A good startup-management mindset is:

- auto-start only the tools you truly use every session
- avoid launching heavy apps automatically unless necessary
- review startup entries after installing new desktop software
- remember that session clutter often feels like system slowness

A useful analogy is to think of startup items as lights and machines that power on when you enter a lab. Some should start immediately. Others should wait until you actually need them.

CLI example: open Startup Applications graphically

```
gnome-session-properties &
```

Example output:

```
[1] 6280
```

This command is directly supported by the Ubuntu Complete Guide.

CLI example: inspect user autostart entries

```
ls ~/.config/autostart
```

Example output:

```
telegram-desktop.desktop
update-notifier.desktop
```

CLI example: create an autostart directory if needed

```
mkdir -p ~/.config/autostart
ls ~/.config
```

Example output:

```
autostart  gtk-3.0  pulse
```

CLI example: inspect active sessions

```
loginctl list-sessions
```

Example output:

```
SESSION  UID USER     SEAT  TTY
      2 1000 student  seat0 tty2
      3 1002 student2 seat0 tty3
```

This helps connect startup behavior and multi-user session behavior to the real system underneath.

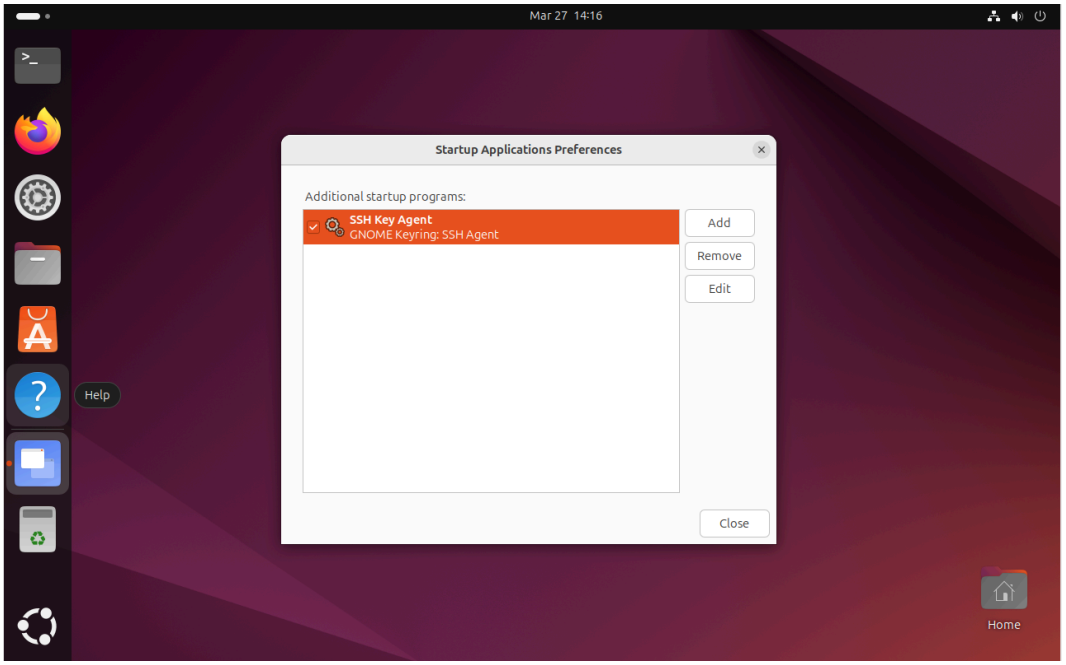


Figure. Startup Applications window.

When to Use GUI Tools and When to Use Terminal

This is one of the most important practical judgment calls in desktop administration. The attached references repeatedly suggest a balanced answer rather than an extreme one. Ubuntu The Complete Guide shows how many common configuration tasks are available through graphical settings, tweak tools, and startup utilities. Ubuntu Linux Bible, on the other hand, stresses that the shell remains the more powerful interface for serious administration and troubleshooting.

A good working guideline is:

Use the **GUI** when:

- the task is visual
- you are teaching beginners
- the setting is easy to verify by sight
- you want low risk and easy discovery

Use the **Terminal** when:

- the task needs exact reproducible commands
- you are writing documentation or lab instructions
- the problem is deeper than the GUI can explain
- you need to search logs, sort processes, or automate a task
- the GUI itself is unstable or unresponsive

This is not a battle between two worlds. It is a partnership. A desktop admin who refuses the GUI loses convenience. A desktop admin who refuses the shell loses power.

A nice illustration is this: the GUI is like a well-labeled workshop wall with switches and knobs. The Terminal is the tool chest. A skilled technician uses both.

CLI example: compare a GUI task and a Terminal task

Open Settings graphically:

```
gnome-control-center &
```

Check the same system through Terminal:

```
hostnamectl
```

Example output:

```
 Static hostname: ubuntu-desktop
       Icon name: computer-laptop
         Chassis: laptop
      Machine ID: 4f0c...
Operating System: Ubuntu 24.04 LTS
          Kernel: Linux 6.8.0-52-generic
    Architecture: x86-64
```

This demonstrates how GUI and CLI often complement the same administrative goal from different angles.

Desktop Support Best Practices for Labs, Schools, and Shared PCs

Ubuntu Desktop is especially attractive for classrooms, schools, labs, and shared machines because it combines a friendly interface with Linux's strong multi-user model. The attached references support this in two important ways. First, Ubuntu's beginner-oriented material emphasizes that Ubuntu is popular in universities and research groups because it offers Unix/Linux power with a customized graphical interface. Second, Ubuntu Complete Guide explicitly notes that Linux was designed from the start to support **multiple users securely**, with separated files and settings, and even guest-style access in some desktop contexts.

For these shared environments, good desktop support should prioritize:

- **separate user accounts** for each person
- limited use of administrator rights
- consistent startup configurations
- safe software installation practices
- routine update checks
- simple recovery and documentation habits
- clean logout and locking behavior
- minimal desktop clutter and predictable defaults

A practical support checklist for labs and shared PCs includes:

- verify network access
- check free disk space
- review running processes if the machine feels slow
- make sure only necessary startup apps are enabled
- document any non-default changes
- train users to lock and log out properly
- preserve privacy between accounts

A useful illustration is to imagine a school computer lab. The best desktop is not the most personalized one. It is the one that is **stable, understandable, recoverable, and respectful of each user's data**.

CLI example: quick shared-PC health check

```
who
df -h
uptime
```

Example output:

```
student  tty2  2026-03-14 09:05
student2 tty3  2026-03-14 09:12

Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p2  236G   31G  194G  14% /
```

```
 10:18:01 up 2:41,  2 users,  load average: 0.52, 0.49, 0.47
```

CLI example: identify current startup items on a shared machine

```
ls ~/.config/autostart
```

Example output:

```
update-notifier.desktop
```

CLI example: launch System Monitor for a support session

```
gnome-system-monitor &
```

Example output:

```
[1] 6401
```

These small routines help administrators spot common issues quickly without making support feel complicated.

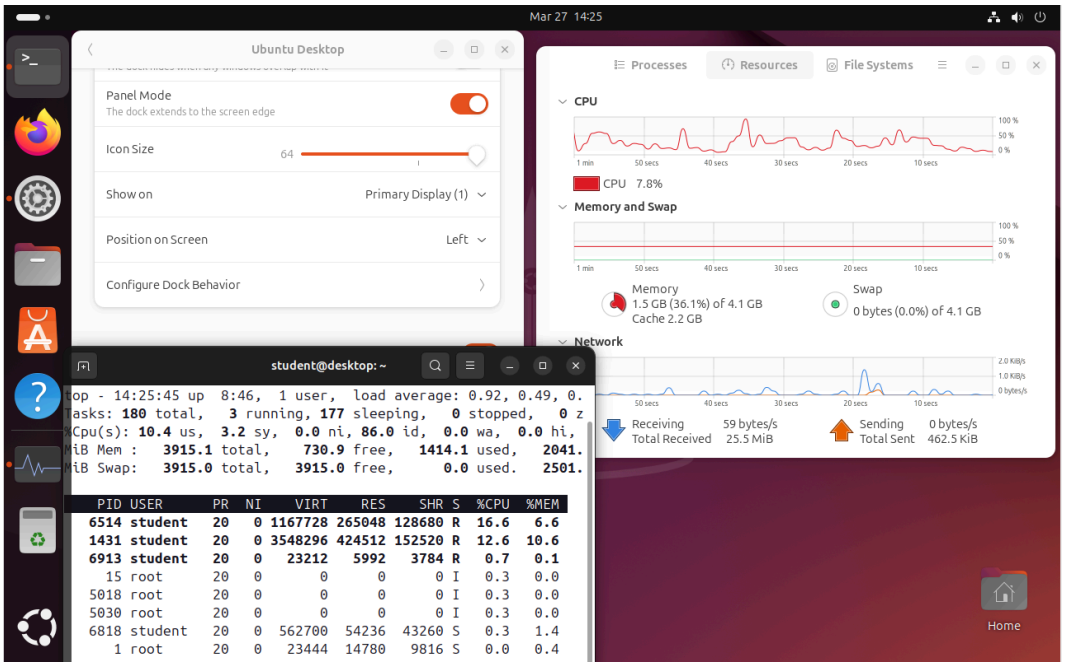


Figure. Workflow view showing Settings, System Monitor, and Terminal open together.

Guided Desktop Administration Lab

To make this chapter hands-on, readers can complete a short support-focused exercise:

- open **Settings** and identify one task better suited to the GUI
- launch **Terminal** and identify one task better suited to the CLI
- run `hostnamectl`, `df -h`, and `uptime`
- open **System Monitor** and sort by CPU or Memory
- run `ps -eo pid,user,group,gid,vsz,rss,comm --sort=-vsz | head`
- start **top** and observe live activity
- open **Startup Applications** with `gnome-session-properties`
- inspect `~/.config/autostart`
- try one `sudo` command safely
- write a short support checklist for a classroom or shared Ubuntu PC

This kind of exercise teaches not only commands, but also the **judgment** of choosing the right administrative tool.

Chapter Reflection

Desktop administration on Ubuntu is most effective when it combines the strengths of the graphical desktop with the precision of the Linux command line. The Settings app and related GNOME tools provide approachable control for common visual and user-facing tasks, while the Terminal unlocks deeper status checks, logs, automation, and reproducible administration. Safe privilege escalation through `sudo` protects the system while still allowing important tasks to be completed. Diagnostic habits grow from a small set of reliable commands and log-reading practices. Process tools such as `ps`, `top`, and **System Monitor** help identify desktop slowdowns and frozen applications. Startup and session management keep login behavior clean and predictable. In shared settings such as labs, schools, and public PCs, these tools become even more important because stability, privacy, and repeatability matter every day.

Key Takeaways

- **Ubuntu Desktop administration uses both GUI and CLI tools effectively.**
- **`sudo` is the safe standard path for temporary administrative access.**
- **Basic diagnostics begin with simple commands and readable logs.**
- **`ps`, `top`, and System Monitor** are essential for process troubleshooting.
- **Startup Applications and session tools** help keep the desktop clean and predictable.
- **The best admins choose the right interface for the task**, not the same interface for every task.
- **Shared PCs in labs and schools benefit from disciplined, repeatable support practices.**

Part VI — Hardware, Devices, and Multimedia

Chapter 13 — Hardware Detection and Driver Management

Ubuntu Desktop works best when it understands the machine beneath it. A smooth graphical session, stable Wi-Fi, clear audio, responsive Bluetooth, good battery life, and reliable external displays all depend on **hardware detection and driver management**. In the Ubuntu Academy outline, this chapter is explicit and practical: **hardware detection tools, GPU drivers, PRIME for hybrid graphics, sound and Bluetooth management, laptop power handling, and external devices** are core desktop-administration skills.

The attached references reinforce this focus from several directions. The Ubuntu Linux Bible explains how commands such as `lspci`, `lsusb`, and `lshw` help identify devices and troubleshoot hardware problems. The Ubuntu Complete Guide gives practical desktop-side examples for Bluetooth pairing and sound testing. The Ubuntu Academy outline adds newer desktop expectations such as **PipeWire/PulseAudio awareness, TLP for laptops, and multi-monitor use**.

A useful illustration is to think of Ubuntu Desktop as an orchestra conductor. The operating system can only create a good performance when it knows which instruments are present, which ones are working, and which ones need special attention. Hardware tools are how Ubuntu learns the orchestra.

Understanding Ubuntu Hardware Support

Ubuntu Desktop is designed to work on a very wide range of hardware. For many laptops and desktops, core devices such as storage, USB ports, displays, keyboards, and wired networking work automatically. That is one of Ubuntu's biggest strengths for everyday users: much of the hardware support is already included in the kernel, firmware packages, and default desktop stack. The Ubuntu Linux Bible emphasizes that users should verify whether their processor, RAM, storage, and boot method are suitable for Ubuntu Desktop, and it recommends `lshw` from a live session as one of the easiest ways to inspect detected hardware before or after installation.

However, "supported" does not always mean "optimized." Some hardware works out of the box but performs better with additional drivers. Some Wi-Fi chipsets, especially certain Broadcom or Realtek models, may need extra packages. Some NVIDIA systems may boot with open drivers first but deliver better 3D performance or screen management after proprietary drivers are installed. That is why good hardware support on Ubuntu is really a **two-stage idea**:

- **detection** — does Ubuntu see the device?
- **optimization** — is Ubuntu using the best driver or configuration for it?

A practical desktop mindset is this: do not assume every issue is a “Linux problem.” Often the system already sees the hardware correctly; the real task is finding the right driver, the right setting, or the right output device.

Detecting Hardware

Before changing any driver, the first job is always to **identify the hardware correctly**. Ubuntu Academy lists four especially useful tools here: `lspci`, `lsusb`, `lshw`, and `inxi`. These are valuable because they answer different questions about the machine.

A helpful way to think about them:

- `lspci` → What PCI and onboard devices are installed?
- `lsusb` → What USB devices are attached?
- `lshw` → What does Ubuntu know about the whole hardware layout?
- `inxi` → Give me a readable summary for support and diagnostics

lspci

The Ubuntu Linux Bible explicitly recommends `lspci` for identifying PCI devices such as graphics cards, USB controllers, audio devices, and Ethernet controllers. It even shows how the output reveals model names and numbers that can then be searched or matched to known driver solutions.

CLI example: list major PCI devices

```
lspci
```

Example output:

```
00:02.0 VGA compatible controller: Intel Corporation UHD Graphics 620
00:14.0 USB controller: Intel Corporation Sunrise Point-LP USB 3.0 xHCI
Controller
00:1f.3 Audio device: Intel Corporation Sunrise Point-LP HD Audio
01:00.0 Network controller: Intel Corporation Wireless 8265 / 8275
```

This is often the fastest first check when graphics, sound, wired networking, or Wi-Fi behave unexpectedly.

CLI example: get more detailed PCI information

```
lspci -vvv
```

Example output:

```
01:00.0 Network controller: Intel Corporation Wireless 8265 / 8275 (rev 78)
        Subsystem: Intel Corporation Dual Band Wireless-AC 8265
        Capabilities: [c8] Power Management version 3
```

The Ubuntu Linux Bible specifically notes that adding `-v` options gives more verbose details, including capabilities and controller information.

lsusb

If the issue involves a USB headset, Bluetooth dongle, webcam, storage device, or printer adapter, `lsusb` is usually the right first command. The Ubuntu Linux Bible directly recommends `lsusb` when the user is specifically interested in USB hardware.

CLI example: list USB devices

```
lsusb
```

Example output:

```
Bus 001 Device 002: ID 8087:0a2b Intel Corp. Bluetooth
wireless interface
Bus 001 Device 004: ID 046d:c31c Logitech, Inc. Keyboard K120
Bus 001 Device 005: ID 0bda:58f4 Realtek Semiconductor Corp.
Integrated Webcam
```

lshw

The Ubuntu Linux Bible recommends `lshw` as an alternative for seeing a more complete listing of detected hardware, especially from a live session or while checking whether a system is suitable for Ubuntu Desktop.

CLI example: summarize hardware

```
sudo lshw -short
```

Example output:

```
H/W path        Device      Class          Description
=====================================================
/0                          system         Laptop
/0/0                        memory         8GiB System Memory
/0/1                        processor      Intel(R) Core(TM) i5-8250U CPU
/0/100/1f.3                 multimedia     Audio device
/0/100/14.3                 network        Wireless interface
/0/100/2                    display        VGA compatible controller
```

This is an excellent command for quickly scanning the machine.

inxi

The Ubuntu Academy outline includes `inxi` because it is often easier for support notes and summaries than raw hardware commands. It is especially useful when readers want a readable overview without combining several tools manually.

CLI example: get a readable system summary

```
inxi -F
```

Example output:

```
System:
  Host: ubuntu-desktop Kernel: 6.8.0-52-generic x86_64
Machine:
  Type: Laptop System: Dell product: Inspiron 5480
CPU:
  Info: Quad Core Intel Core i5-8265U
Graphics:
  Device-1: Intel UHD Graphics 620
  Device-2: NVIDIA GeForce MX130
Audio:
  Device-1: Intel Cannon Point-LP High Definition Audio
Network:
  Device-1: Intel Wireless 9560
Drives:
  Local Storage: 256 GiB SSD
```

This is the kind of report many desktop support staff appreciate because it compresses a lot of information into one structured output.

Screenshot notes:
[Screenshot-CH13-02: Terminal showing lspci output for graphics, audio, and network devices.]
[Screenshot-CH13-03: Terminal showing sudo lshw -short on a laptop.]
[Screenshot-CH13-04: Terminal showing inxi -F summary report.]

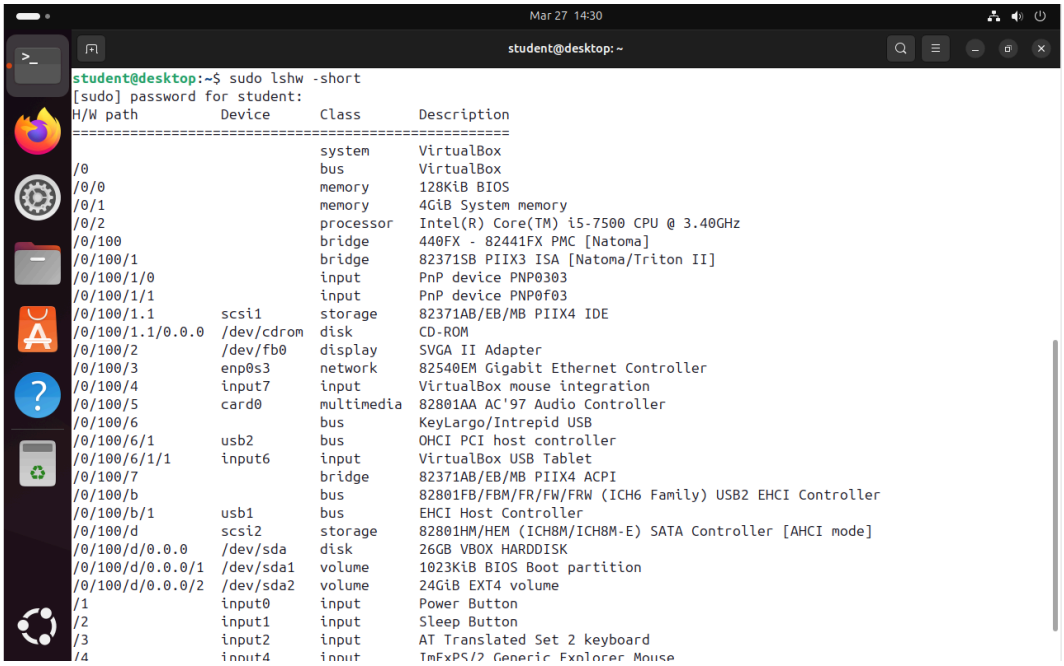


Figure. Terminal showing sudo lshw -short

Graphics Hardware and GPU Drivers

Graphics are among the most important parts of Ubuntu Desktop because they influence the login screen, GNOME animations, video playback, external displays, Wayland/X.Org behavior, and application acceleration. The Ubuntu Academy outline explicitly highlights **proprietary vs open graphics drivers** and mentions **Nouveau** and GPU management directly.

The Ubuntu Linux Bible provides a strong explanation of how `lspci` output identifies graphics hardware and how those model names help determine what driver path to take. The references also distinguish between **open drivers that work immediately** and **proprietary drivers that may be better for performance or compatibility** in certain cases.

A practical desktop summary is:

- **Intel graphics** usually work very well with the built-in open stack
- **AMD graphics** often work well with open drivers and modern Mesa support
- **NVIDIA graphics** may work with open drivers such as Nouveau, but many users prefer proprietary drivers for stronger performance or fewer compatibility issues in certain workloads

A useful illustration is to think of GPU drivers as translators between the desktop and the graphics hardware. A basic translator may already allow the conversation to happen. A better one may make the conversation faster and more accurate.

CLI example: detect graphics hardware

```
lspci | grep -i vga
```

Example output:

```
00:02.0 VGA compatible controller: Intel Corporation UHD Graphics 620
01:00.0 VGA compatible controller: NVIDIA Corporation GP108M [GeForce MX150]
```

CLI example: ask Ubuntu what drivers are available

```
sudo ubuntu-drivers devices
```

Example output:

```
== /sys/devices/pci0000:00/0000:00:02.0 ==
vendor   : Intel Corporation
model    : UHD Graphics 620

== /sys/devices/pci0000:00/0000:01:00.0 ==
vendor   : NVIDIA Corporation
model    : GP108M [GeForce MX150]
driver   : nvidia-driver-535 - third-party non-free recommended
driver   : xserver-xorg-video-nouveau - distro free builtin
```

This is especially helpful for NVIDIA systems because it clearly shows the open and recommended proprietary paths.

CLI example: install recommended drivers automatically

```
sudo ubuntu-drivers autoinstall
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  nvidia-driver-535 nvidia-dkms-535
Done.
```

CLI example: verify the NVIDIA driver

```
nvidia-smi
```

Example output:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.171.04    Driver Version: 535.171.04    CUDA Version: 12.2   |
+-----------------------------------------------------------------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A |
|  0   GeForce MX150      Off   | 00000000:01:00.0 Off |
+-----------------------------------------------------------------------------+
```

For AMD systems, the focus is often on verifying rendering rather than installing a separate proprietary package:

```
glxinfo | grep -i render
```

Example output:

```
direct rendering: Yes
OpenGL renderer string: AMD Radeon RX 560X (RADV POLARIS11)
```

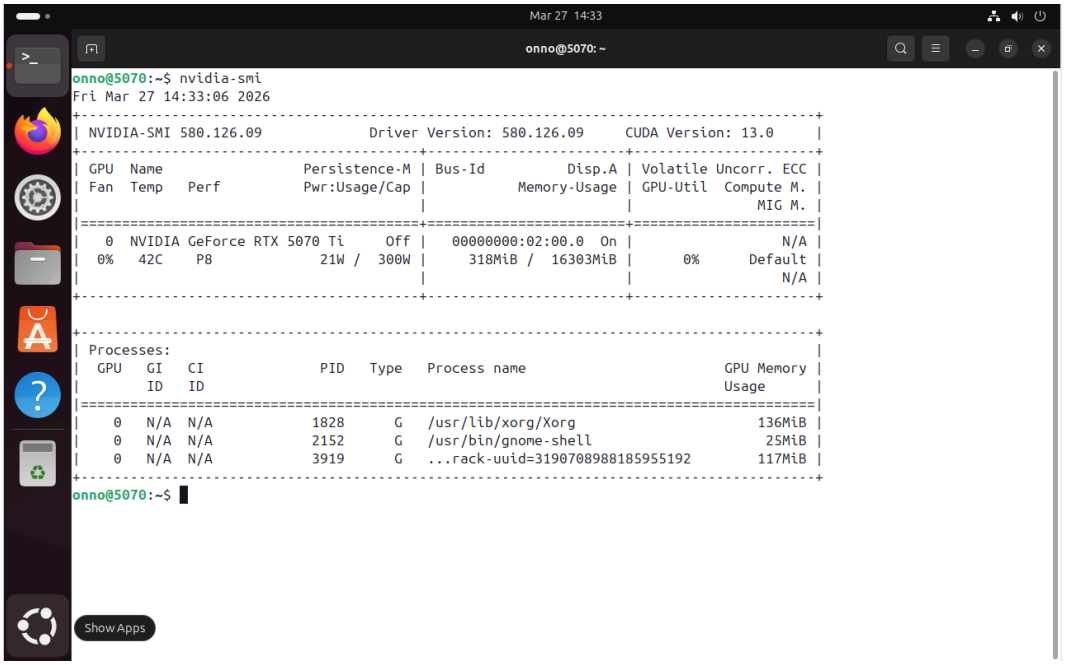


Figure. nvidia-smi output on a PC with proprietary NVIDIA driver installed.

Hybrid Graphics and PRIME Concepts

Many modern laptops include **hybrid graphics**, where an integrated GPU and a discrete GPU are both present. The Ubuntu Academy outline explicitly names **PRIME for hybrid graphics**, which is especially relevant on laptops that combine Intel or AMD integrated graphics with NVIDIA discrete graphics.

The main idea behind PRIME is that the system can choose which GPU handles rendering depending on performance, battery needs, and session configuration. This matters because a laptop may need one of two priorities:

- **battery life and cooler operation**
- **higher graphics performance**

A useful illustration is to think of hybrid graphics like a car with eco mode and sport mode. One mode favors efficiency. The other favors power.

CLI example: see available PRIME profiles

```
sudo prime-select query
```

Example output:

```
on-demand
```

Possible outputs may include:

```
nvidia
```

or

```
intel
```

depending on system generation and driver stack.

CLI example: switch to on-demand mode

```
sudo prime-select on-demand
```

Typical result:

```
Info: selecting the on-demand profile
```

CLI example: switch to NVIDIA-only mode

```
sudo prime-select nvidia
```

Typical result:

```
Info: selecting the nvidia profile
```

A reboot or new login session is often required after changing PRIME mode.

This topic is important because users sometimes assume poor battery life or odd multi-display behavior is a “desktop problem,” when the real cause is the current GPU mode.

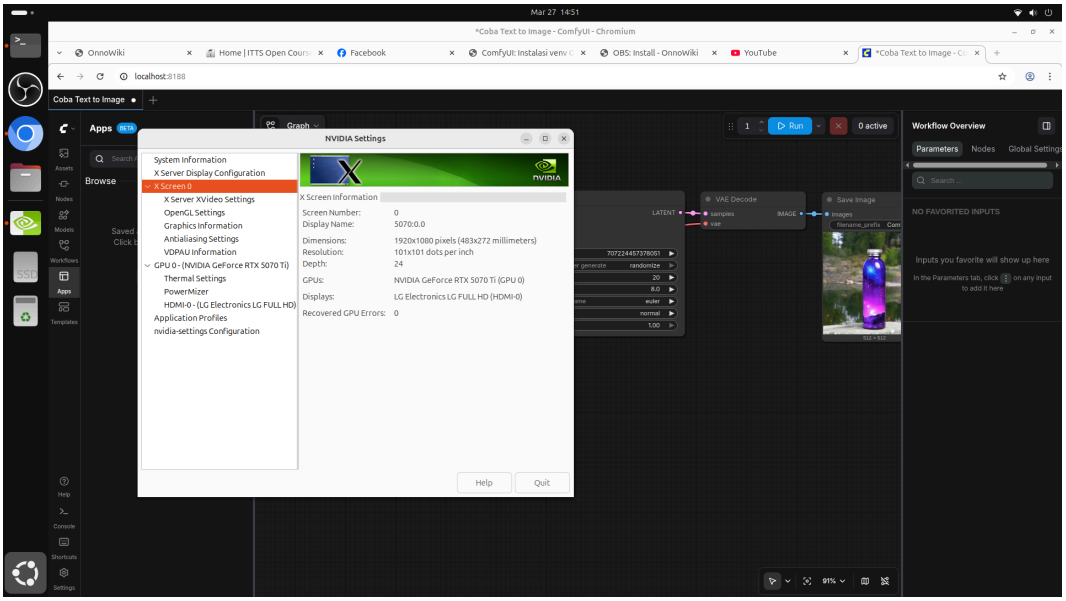


Figure. NVIDIA Settings.

Audio Devices and Troubleshooting Basics

Good sound support is essential on a desktop. The Ubuntu Academy outline explicitly includes **PulseAudio/PipeWire management** and sound troubleshooting, while the Ubuntu Complete Guide provides very practical examples for both **graphical sound testing** and **command-line diagnostics**. It explains that the Sound Settings interface includes a **Test Sound** function, recommends `aplay /usr/share/sounds/alsa/Front_Center.wav` as a quick speaker test, explains the usefulness of `alsamixer`, and even shows how to reset user PulseAudio settings when necessary.

A strong beginner troubleshooting order is:

- verify the correct **output device**
- test volume and mute state
- test with a known-good sound file
- inspect mixer levels
- try a second device such as speakers or headphones
- restart or reset user-side sound configuration only if needed

A useful analogy is to think of sound troubleshooting like checking a classroom microphone: first confirm the correct speaker, then volume, then cable or wireless route, then the mixer.

CLI example: test basic sound playback

```
aplay /usr/share/sounds/alsa/Front_Center.wav
```

Example output:

```
Playing WAVE '/usr/share/sounds/alsa/Front_Center.wav' :
Signed 16 bit Little Endian, Rate 48000 Hz, Mono
```

The Ubuntu Complete Guide explicitly recommends this exact sound test.

CLI example: open ALSA mixer

```
alsamixer
```

Example output:

```
(terminal opens interactive mixer with playback devices
and volume bars)
```

The Ubuntu Complete Guide specifically recommends `alsamixer` for adjusting playback devices with the keyboard.

CLI example: list audio devices

```
aplay -l
```

Example output:

```
card 0: PCH [HDA Intel PCH], device 0: ALC256 Analog
[ALC256 Analog]
card 1: HDMI [HDA NVIDIA HDMI], device 3: HDMI 0 [HDMI
0]
```

CLI example: reset user PulseAudio settings

```
sudo rm -r ~/.config/pulse; pulseaudio -k
```

Typical result:

```
(no output if successful)
```

The Ubuntu Complete Guide explicitly gives this command as a last-step reset for user-side sound settings.

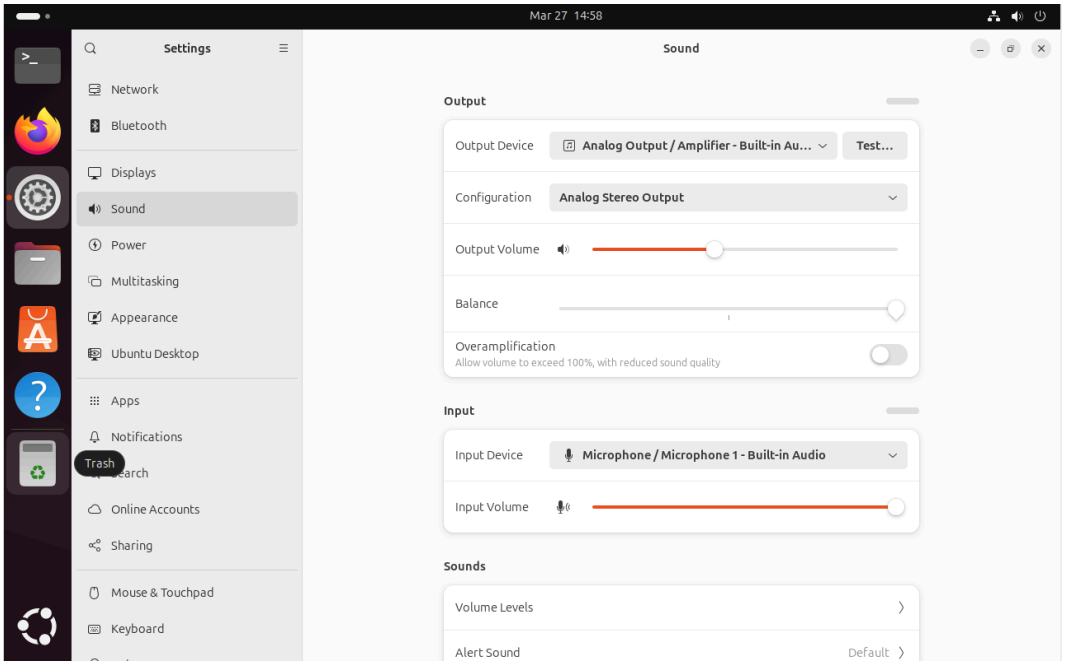


Figure. Settings > Sound showing output device selection and Test Sound button.

Bluetooth Devices and Pairing

Bluetooth is one of the most common desktop convenience features for headphones, speakers, keyboards, mice, and mobile accessories. The Ubuntu Academy outline includes Bluetooth pairing directly, and the Ubuntu Complete Guide provides a clear pairing workflow for **Bluetooth speakers**. It explains that if Ubuntu does not support Bluetooth hardware, the search result may show **No Bluetooth adapters found**, and it walks through enabling Bluetooth, making the computer discoverable, pressing the **+** button to search, and then selecting the new device as the sound output.

A practical Bluetooth workflow is:

- confirm the machine has a Bluetooth adapter
- turn Bluetooth on
- put the target device into pairing or discovery mode
- pair the device in the Bluetooth settings
- if it is an audio device, confirm it is selected in Sound Settings

A good illustration is to think of Bluetooth pairing like introducing two people at a meeting: both sides must be present, visible, and ready to connect.

CLI example: check for USB Bluetooth hardware

```
lsusb | grep -i bluetooth
```

Example output:

```
Bus 001 Device 002: ID 8087:0a2b Intel Corp. Bluetooth
wireless interface
```

CLI example: show Bluetooth service status

```
systemctl status bluetooth
```

Example output:

```
● bluetooth.service - Bluetooth service
     Loaded: loaded (/lib/systemd/system/bluetooth.service; enabled)
     Active: active (running) since Fri 2026-03-14 08:12:04 WIB
```

CLI example: open Bluetooth control in the desktop

```
gnome-control-center bluetooth &
```

Example output:

```
[1] 6774
```

These commands complement the graphical pairing process described in the Ubuntu Complete Guide.

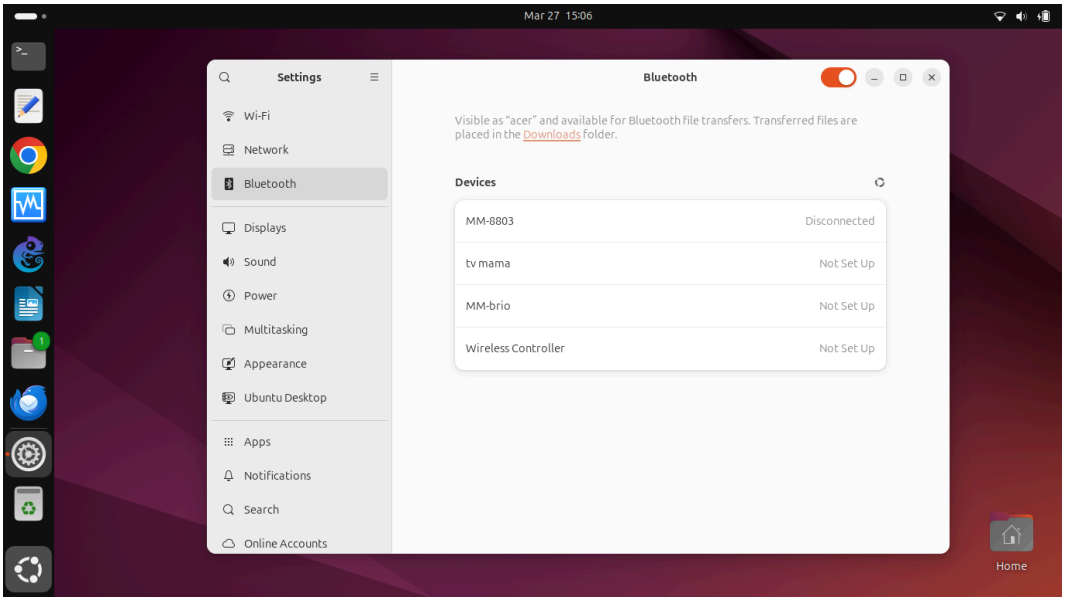


Figure. Bluetooth pairing window with a headset or speaker found and ready to pair

Power and Battery Management for Laptops

Laptop users experience Ubuntu through battery life, suspend behavior, fan noise, brightness, and thermal comfort just as much as through applications. The Ubuntu Academy outline explicitly includes **battery settings** and mentions **TLP for laptops** as a useful power-management tool.

The attached references also support the broader principle that desktop choice, configuration, and extensions affect power usage, especially on portable devices. Mastering Ubuntu notes that battery behavior and power consumption are significant considerations for laptop use, even when discussing alternative desktop environments and their different resource profiles.

A practical laptop power strategy usually includes:

- using the correct **Power** settings in the GUI
- reducing brightness when possible
- preferring integrated graphics or on-demand graphics if battery life matters
- limiting unnecessary startup items or heavy background apps
- optionally using tools such as **TLP** for more aggressive laptop optimization

A useful illustration is to think of battery management like budgeting energy for a long trip. Every background service, high-brightness setting, or always-on GPU is part of the spending plan.

CLI example: inspect power devices

```
upower -e
```

Example output:

```
/org/freedesktop/UPower/devices/battery_BAT0
/org/freedesktop/UPower/devices/line_power_AC
```

CLI example: show detailed battery information

```
upower -i /org/freedesktop/UPower/devices/battery_BAT0
```

Example output:

```
 native-path:          BAT0
  power supply:         yes
  updated:              Fri 14 Mar 2026 09:41:12 AM WIB
  percentage:           84%
  state:                discharging
  time to empty:        3.8 hours
```

CLI example: install TLP

```
sudo apt install tlp
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  tlp
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
```

CLI example: check TLP status

```
sudo tlp-stat -s
```

Example output:

```
--- TLP 1.6 ------------------------------
System state = battery
TLP power save = enabled
```

This gives laptop users a practical power-management checkpoint that goes beyond the graphical battery icon.

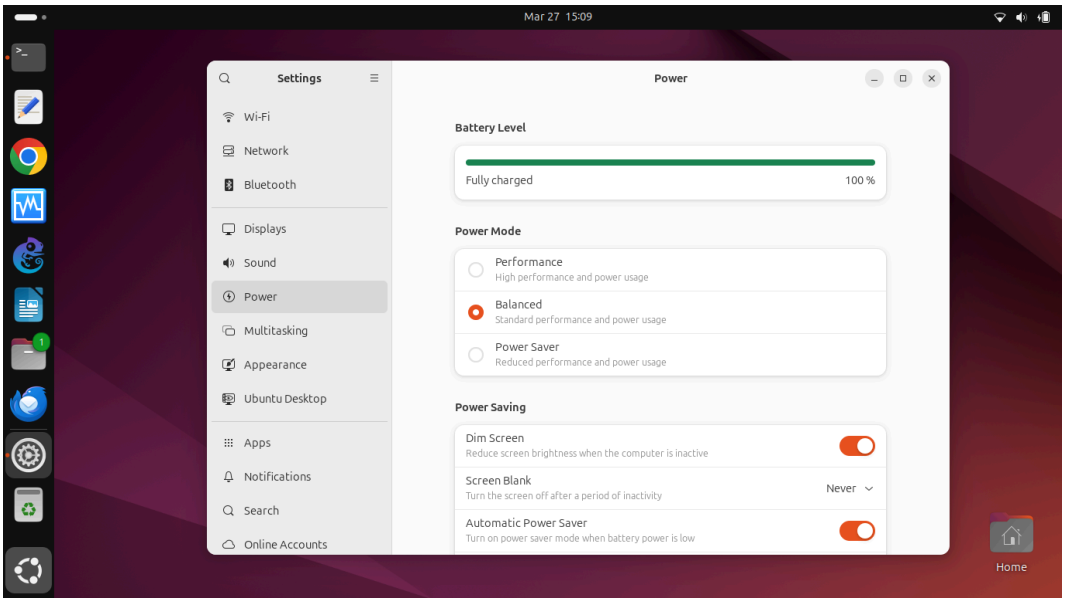


Figure. Settings > Power showing battery mode, screen blanking, and suspend settings.

External Displays, Docking, and Multi-Monitor Setups

Ubuntu Desktop is often used with projectors, office monitors, classroom display walls, USB-C docks, and multi-screen workstations. The Ubuntu Academy outline explicitly includes **external displays, docking, and multi-monitor setups**, and the Ubuntu Complete Guide highlights **Display settings** and practical desktop display configuration as part of Ubuntu system settings.

Display management on Ubuntu usually involves:

- detecting whether the external display is seen
- choosing mirror or extended mode
- arranging monitors in the correct physical order
- selecting the main display
- adjusting resolution and scaling
- confirming audio routing if HDMI/DisplayPort audio is involved

A useful illustration is to think of multi-monitor configuration as arranging multiple whiteboards in a room. It is not enough that they exist; they must be placed in the right order so movement across them feels natural.

CLI example: list display outputs with xrandr

```
xrandr
```

Example output:

```
Screen 0: minimum 8 x 8, current 3840 x 1080, maximum 32767 x 32767
eDP-1 connected primary 1920x1080+0+0
HDMI-1 connected 1920x1080+1920+0
```

CLI example: enable an external monitor to the right

```
xrandr --output HDMI-1 --auto --right-of eDP-1
```

Typical result:

```
(no output if successful)
```

CLI example: mirror the laptop screen

```
xrandr --output HDMI-1 --same-as eDP-1
```

Typical result:

```
(no output if successful)
```

Even when GNOME Settings is the preferred tool for most users, `xrandr` remains a valuable diagnostic and emergency tool, especially when troubleshooting docks, classroom projectors, or unusual monitor arrangements.

Screenshot notes:

[Screenshot-CH13-17: Settings > Displays showing a laptop display and an external monitor arranged side by side.]
[Screenshot-CH13-18: Multi-monitor Ubuntu desktop with browser on one screen and document editor on the other.]
[Screenshot-CH13-19: External projector or classroom display connected to an Ubuntu laptop.]

Guided Hardware and Driver Lab

To make this chapter hands-on, readers can complete a practical hardware-detection activity:

- run `lspci`, `lsusb`, `sudo lshw -short`, and `inxi -F`
- identify the system's **graphics**, **audio**, **network**, and **Bluetooth** hardware
- run `sudo ubuntu-drivers devices` on a system with a discrete GPU
- test audio with `aplay /usr/share/sounds/alsa/Front_Center.wav`
- open `alsamixer` and verify the correct output path
- pair a Bluetooth headset or speaker
- inspect battery state with `upower -i`
- connect an external monitor and check layout in **Settings > Displays**
- use `xrandr` to verify output names and active resolutions

This kind of exercise helps students move from passive hardware use to active hardware understanding.

Chapter Reflection

Hardware detection and driver management are at the center of a healthy Ubuntu Desktop experience. Ubuntu usually supports a wide range of devices automatically, but reliable desktop administration depends on knowing how to verify what the system sees and how it is configured. Tools such as `lspci`, `lsusb`, `lshw`, and `inxi` reveal different layers of hardware identity. Graphics management requires understanding the difference between open and proprietary drivers, especially for NVIDIA and hybrid-graphics laptops using PRIME. Sound troubleshooting begins with the correct device, test playback, and mixer inspection. Bluetooth pairing depends on both adapter detection and correct output selection. Laptop battery life benefits from good power settings and tools such as TLP. External displays and docking setups become much easier when users understand detection, arrangement, and display-output naming. Together, these tools help Ubuntu Desktop feel less mysterious and much more controllable.

Key Takeaways

- **Ubuntu hardware management begins with accurate detection, not guesswork.**
- **`lspci`, `lsusb`, `lshw`, and `inxi`** each answer different hardware questions.
- **Graphics drivers matter most on NVIDIA and hybrid-graphics systems**, where proprietary drivers and PRIME may improve results.
- **Sound troubleshooting should start with output selection, sound testing, and mixer inspection.**
- **Bluetooth pairing is a two-step problem: adapter detection and device pairing/output selection.**
- **Laptop power management benefits from both GUI settings and tools like TLP.**
- **External displays become easier to manage when users understand monitor detection and layout tools.**

Chapter 14 — Printers, Scanners, Storage, and Removable Media

A desktop computer is not useful only because of what appears on the screen. It is also useful because of the devices it can work with: **printers, scanners, USB drives, DVDs, memory cards, and external storage**. In the Ubuntu Academy outline, this chapter focuses on **CUPS printing, scanners, removable media, automount behavior, filesystems, mount points, and troubleshooting peripheral detection**. That focus is very practical because these are exactly the kinds of tasks users meet in schools, labs, offices, and everyday home computing.

The attached references strongly support this chapter from two directions. The **Ubuntu Linux Bible** explains that Linux hardware handling has become more automatic thanks to the kernel and the **Udev** subsystem, which dynamically creates and manages devices as hardware appears and disappears. It also explains that removable media may be mounted automatically and that tools such as `mount`, `dmesg`, `journalctl`, and `tail -f /var/log/syslog` help investigate hardware behavior. Meanwhile, **Ubuntu: The Complete Guide** gives very practical desktop guidance for flash drives, optical media, and safe unmounting.

A useful illustration is to think of Ubuntu Desktop as a busy office reception desk. Printers are the outgoing document channel, scanners are the input channel, and removable media are like visitors bringing or taking files. A good desktop administrator needs to know how these channels connect, how they are mounted into the system, and how to troubleshoot them when something goes wrong.

Printer Setup with CUPS

Printing on Ubuntu is built around **CUPS**, the **Common Unix Printing System**. The Ubuntu Linux Bible states this directly and explains that Ubuntu offers several ways to manage printing: a graphical **Printers** window, configuration files under `/etc/cups`, and even a web-based administrative interface for CUPS. It also notes that packages such as `cups`, `system-config-printer`, and printer-driver packages make it possible to support many different printer types, including printers shared on a network.

This is an important desktop lesson: printing on Ubuntu is not only “plug in printer and hope.” It is a full service architecture. CUPS acts like a print traffic manager. It receives print jobs, queues them, sends them to the correct printer, and tracks printer status.

A practical beginner mental model is:

- **CUPS** = the print management system
- **Printers window** = the graphical control panel
- **drivers** = the translation layer between Ubuntu and the printer
- **print queue** = the waiting line for print jobs

A good analogy is to imagine CUPS as the school print room. It knows which printer is available, which jobs are waiting, and which documents should go to which device.

CLI example: verify that CUPS is installed

```
dpkg -l | grep cups
```

Example output:

```
ii  cups           2.4.7-1ubuntu2   amd64   Common UNIX Printing System(tm) - PPD/driver support
ii  cups-daemon    2.4.7-1ubuntu2   amd64   Common UNIX Printing System(tm) - daemon
ii  cups-client    2.4.7-1ubuntu2   amd64   Common UNIX Printing System(tm) - client programs
```

CLI example: check CUPS service status

```
systemctl status cups
```

Example output:

```
● cups.service - CUPS Scheduler
     Loaded: loaded (/lib/systemd/system/cups.service; enabled)
     Active: active (running) since Fri 2026-03-14 08:15:12 WIB
```

CLI example: install graphical printer tools if needed

```
sudo apt install cups system-config-printer
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  cups system-config-printer
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
```

The Ubuntu Linux Bible explicitly mentions the **system-config-printer** package as the graphical tool for printer administration.

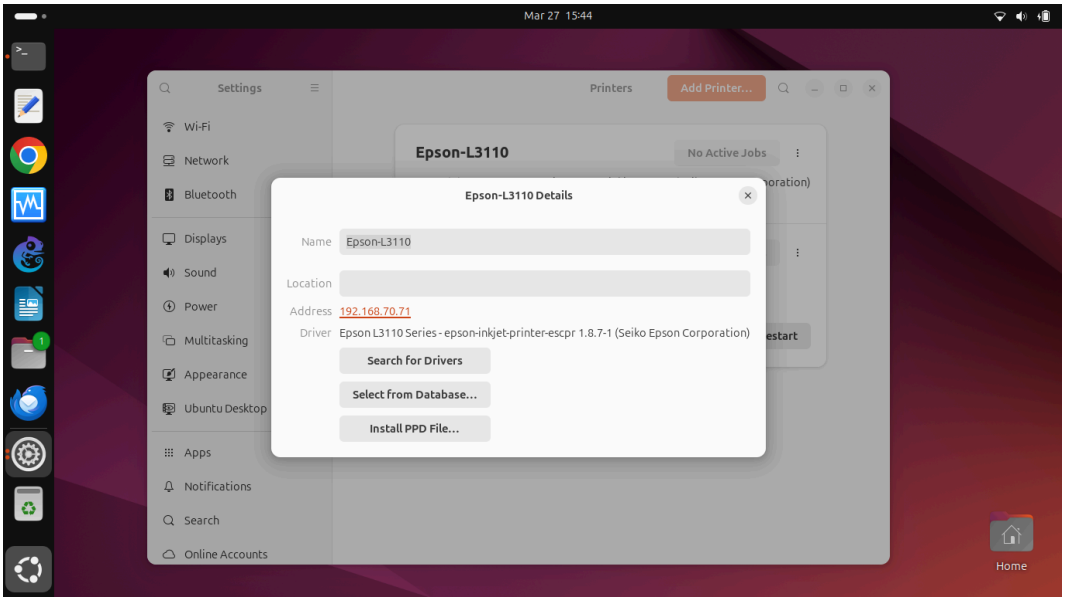


Figure. Settings or Printers window showing an installed network printer

Adding and Managing Printers

The Ubuntu Linux Bible describes printer setup in a very practical way: once the printer is detected, Ubuntu can guide the user through an **Add Printer Wizard**, ask how the printer is connected, and let the user select drivers. It also gives hands-on exercises using the **Printers window**, `lpstat -t`, `lp`, and `cancel` to manage printer status and jobs.

This supports a clear desktop workflow:

- connect a local USB printer or identify a network printer
- open the **Printers** window
- add the printer
- choose the correct driver or generic driver if necessary
- print a test page
- review the printer queue and status

The references also note that Ubuntu can publish printers to the local network and even emulate SMB/Windows-style print sharing when needed. That is especially relevant in schools and offices where one device may serve many users.

A useful illustration is to think of adding a printer like adding a new staff member to the office: Ubuntu needs to know the device's identity, its role, and how to communicate with it before work can begin smoothly.

CLI example: show the status of all printers

```
lpstat -t
```

Example output:

```
scheduler is running
system default destination: OfficePrinter
device for OfficePrinter: usb://HP/LaserJet_Pro_M404
OfficePrinter accepting requests since Fri 14 Mar 2026 09:12:14 WIB
printer OfficePrinter is idle. enabled since Fri 14 Mar 2026 09:12:14 WIB
```

The Ubuntu Linux Bible explicitly recommends `lpstat -t` for seeing the status of all printers.

CLI example: print a test file

```
lp /etc/hosts
```

Example output:

```
request id is OfficePrinter-12 (1 file(s))
```

This exact exercise pattern appears in the Ubuntu Linux Bible, using `lp` to print `/etc/hosts`.

CLI example: cancel a print job

```
cancel OfficePrinter-12
```

Typical result:

```
(no output if successful)
```

CLI example: open the printer management window

```
system-config-printer &
```

Example output:

```
[1] 6891
```

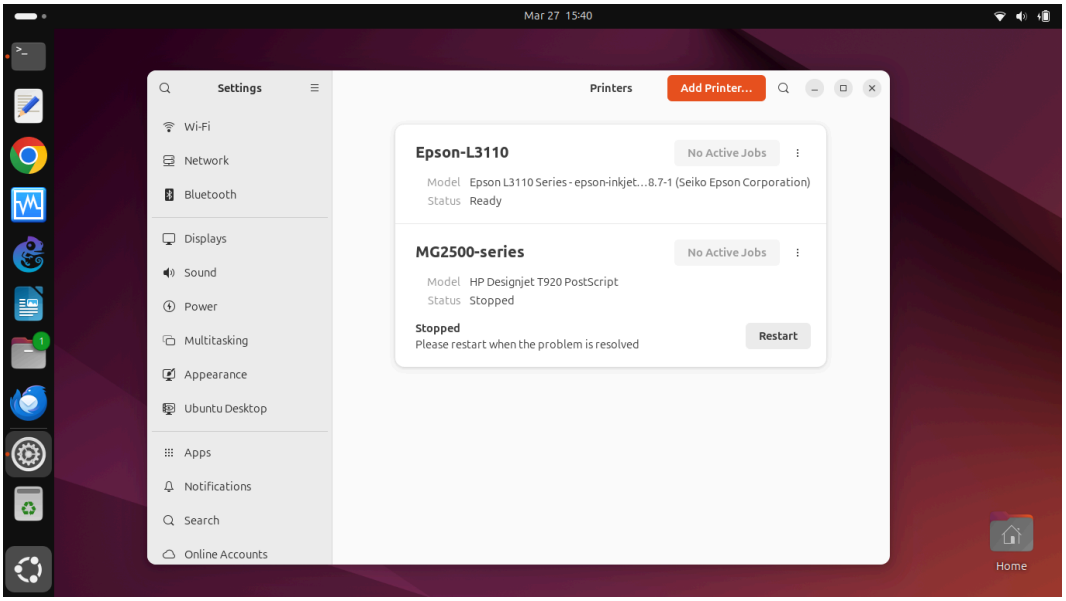


Figure. Add Printer Wizard selecting a USB or network printer.

Scanners and Document Capture

Modern multifunction devices are often not just printers. The Ubuntu Linux Bible explicitly notes that Linux should be able to look at the same hardware in different ways—for example, a device may be recognized as a **printer, fax machine, scanner, and storage device** at the same time. This is a very important idea for Ubuntu Desktop users because many office peripherals are multifunction devices.

The first task in scanner support is not to guess. It is to verify whether Ubuntu sees the device at all. If the scanner is part of a USB multifunction printer, `lsusb`, `dmesg`, and desktop scanner tools can help confirm whether the scanner side is available.

A good scanner workflow usually includes:

- connect the device
- verify that Ubuntu detects it
- open a document-scanning application
- choose the correct device
- select document or photo mode
- scan and save in a well-organized folder

A simple illustration is to think of a scanner as the inverse of a printer. A printer turns digital work into paper. A scanner brings paper back into the digital workspace.

CLI example: inspect USB devices for scanner or multifunction hardware

```
lsusb
```

Example output:

```
Bus 001 Device 006: ID 03f0:3f17 HP, Inc. LaserJet Pro MFP M428fdw
Bus 001 Device 004: ID 046d:c52b Logitech, Inc. Unifying Receiver
```

CLI example: follow kernel messages while plugging in a scanner

```
journalctl -f
```

Example output:

```
Mar 14 10:22:11 ubuntu-desktop kernel: usb 1-2: new
high-speed USB device number 6 using xhci_hcd
Mar 14 10:22:11 ubuntu-desktop kernel: usb 1-2: New USB
device found, idVendor=03f0, idProduct=3f17
Mar 14 10:22:12 ubuntu-desktop systemd[1]: Started
scanner detection service.
```

The Ubuntu Linux Bible specifically recommends `journalctl -f` and also explains that `dmesg` can reveal what hardware was detected and which drivers were loaded.

CLI example: inspect recent kernel messages

```
dmesg | tail
```

Example output:

```
[ 5123.214011] usb 1-2: new high-speed USB device number
6 using xhci_hcd
[ 5123.348941] usb 1-2: New USB device found,
idVendor=03f0, idProduct=3f17
[ 5123.349102] usblp 1-2:1.0: usblp0: USB Bidirectional
printer dev 6 if 0 alt 1 proto 2 vid 0x03F0 pid 0x3F17
```

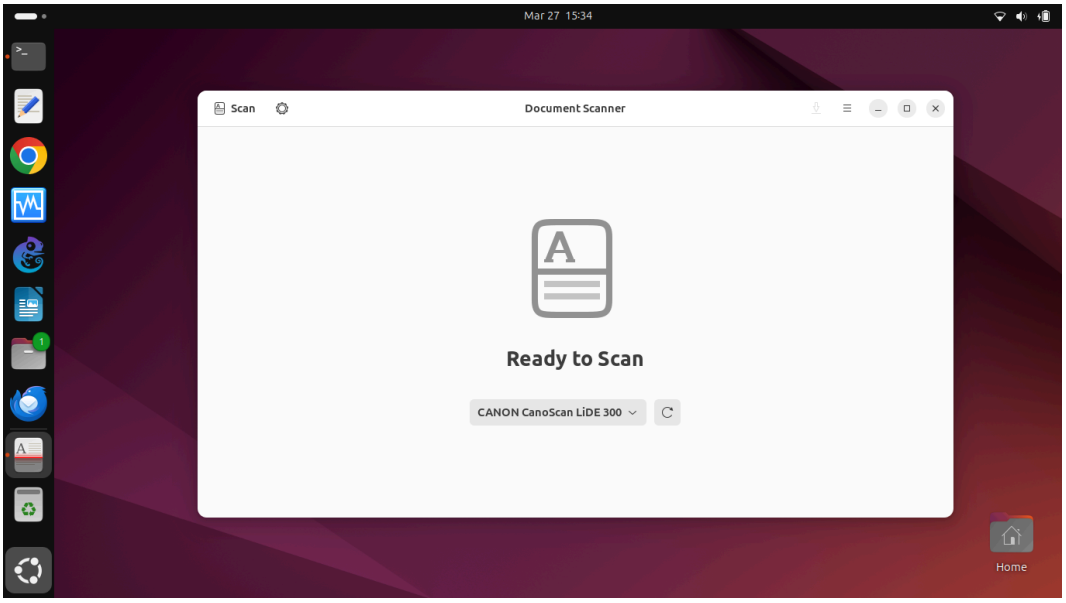


Figure. Document Scanner application open with preview pane and scan settings.

USB Devices and Automount Behavior

The Ubuntu Linux Bible explains that Linux device handling has become much more automatic over time. Thanks to the **Udev** subsystem, devices are detected dynamically as they are connected or removed, interfaces are created, and desktop applications can respond automatically. The book explicitly says that on desktop systems, when hardware such as removable media is inserted, applications may be ready to **mount the hardware and/or launch an application**.

Ubuntu: The Complete Guide gives a very practical desktop description of this behavior. It says that when you plug in a USB flash drive or hard drive, Ubuntu will usually **automatically mount** it and make it accessible, as long as the filesystem is supported. It also explains that Ubuntu supports common filesystems such as **FAT32**, **NTFS**, and its own ext-based filesystems, and it notes that **exFAT** support may need to be added separately in some cases.

This makes automount easy to understand:

- connect the device
- Ubuntu detects it
- a device entry appears in Files or the sidebar
- the filesystem is mounted under a mount point
- the user can access the files immediately

A helpful analogy is to think of automounting as a receptionist preparing a visitor badge and opening the correct meeting room as soon as the visitor arrives.

CLI example: inspect block devices before and after inserting a USB drive

```
lsblk
```

Example output:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 238.5G  0 disk
├─sda1   8:1    0   512M  0 part /boot/efi
└─sda2   8:2    0   238G  0 part /
sdb      8:16   1  29.3G  0 disk
└─sdb1   8:17   1  29.3G  0 part /media/student/USB32
```

CLI example: list all currently mounted filesystems

```
mount
```

Example output:

```
/dev/sda2 on / type ext4 (rw,relatime)
/dev/sda1 on /boot/efi type vfat (rw,relatime)
/dev/sdb1 on /media/student/USB32 type vfat (rw,nosuid,nodev,relatime)
```

The Ubuntu Linux Bible explicitly explains that running `mount` with no options shows currently mounted filesystems and their mount points.

CLI example: follow log messages when inserting a USB drive

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

Example output:

```
Mar 14 10:35:18 ubuntu-desktop kernel: usb 1-3: new
SuperSpeed USB device number 7 using xhci_hcd
Mar 14 10:35:19 ubuntu-desktop udisksd[913]: Mounted
/dev/sdb1 at /media/student/USB32
```

The Ubuntu Linux Bible explicitly recommends `tail -f /var/log/syslog` or `journalctl -f` to observe what happens when a USB device is plugged in.

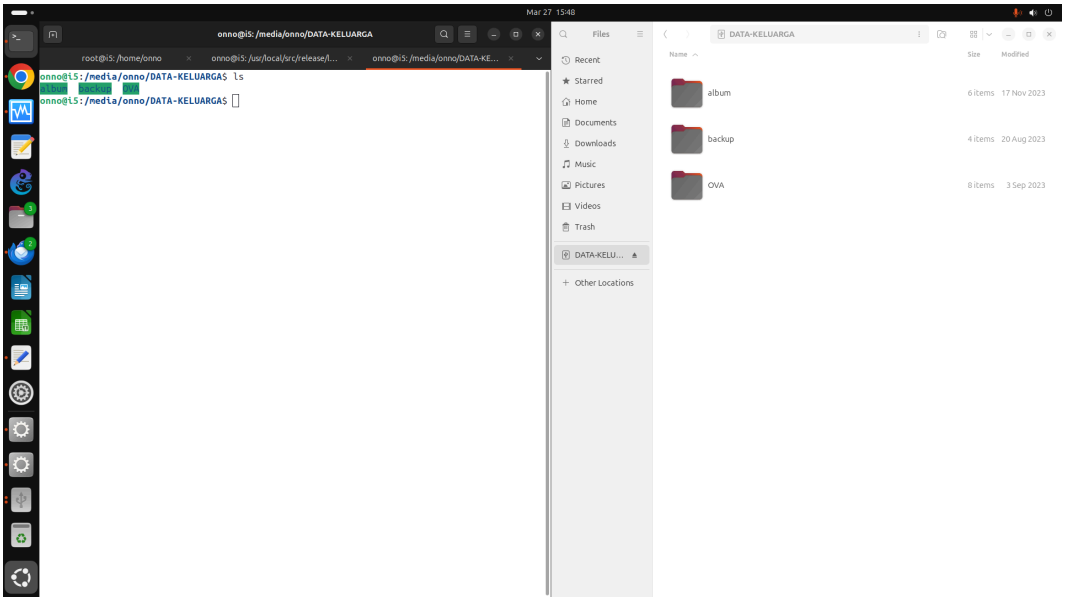


Figure. Files window open to an automounted USB drive under /media/<user>/...

Optical Media and Removable Drives

Although many modern systems rely more on USB media than DVDs or CDs, optical media still appears in classrooms, archives, and older environments. Ubuntu: The Complete Guide specifically includes a section called **Drives: Access flash and optical media**, explaining that Linux handles drives differently from Windows, but that Ubuntu can work comfortably with both flash storage and optical media using built-in tools. It also notes that when optical media is inserted, Ubuntu may show a helpful dialog and launch relevant actions automatically.

The Ubuntu Linux Bible also notes that optical media such as CDs and DVDs are among the most traditional examples of removable media that may be mounted automatically, though users may sometimes still mount them manually. It even shows an example where `/dev/cdrom` is mounted on a path under `/media` with the `iso9660` filesystem.

A practical understanding is:

- optical discs are still removable media
- they may appear under `/media/...`
- the filesystem is often read-only
- music or photo applications may launch automatically depending on content

A useful analogy is to think of an optical disc like a printed archive folder. It may not be rewritten easily, but it can still be opened, browsed, and copied from.

CLI example: inspect mounted optical media

```
mount | grep cdrom
```

Example output:

```
/dev/cdrom on /media/student/MyOwnDVD type iso9660
(ro,nosuid,nodev)
```

This closely follows the example style shown in the Ubuntu Linux Bible.

CLI example: check optical drive capabilities

```
wodim -prcap
```

Example output:

```
Device was not specified. Trying to find an appropriate drive...
Using dev='/dev/sr0'
Mounted media class: Medium Error
Supported modes: TAO SAO RAW/R96R
```

Ubuntu: The Complete Guide explicitly recommends `wodim -prcap` as a quick way to verify what a CD/DVD drive is capable of doing in Ubuntu.

Filesystems and Mount Points in Desktop Context

To use removable media confidently, desktop users need at least a simple understanding of **filesystems** and **mount points**. Ubuntu: The Complete Guide explains that Ubuntu can read and write common filesystems such as **FAT32**, **NTFS**, and ext-based filesystems, and that exFAT may need extra support. The Ubuntu Linux Bible explains the `mount` command and emphasizes that the output reveals both the device and the directory where the filesystem is attached.

This is one of the most important desktop concepts that beginners often miss. A USB drive does not become useful simply because it exists as `/dev/sdb1`. It becomes useful when its filesystem is **mounted** into the directory tree, such as:

- `/media/student/USB32`
- `/media/student/MyOwnDVD`
- another temporary mount point chosen manually

A good beginner illustration is this: the storage device is the bookshelf, but the **mount point is the door** through which Ubuntu lets you enter that shelf and read its contents.

CLI example: show filesystems and mount points

```
lsblk -f
```

Example output:

```
NAME   FSTYPE LABEL   UUID                   MOUNTPOINTS
sda
├─sda1 vfat   EFI     1234-ABCD              /boot/efi
└─sda2 ext4           a1b2c3d4-e5f6-7788-99aa /
sdb
└─sdb1 vfat   USB32   4D3A-9B7C              /media/student/USB32
```

CLI example: mount a filesystem manually

```
sudo mkdir -p /mnt/usbtest
sudo mount /dev/sdb1 /mnt/usbtest
mount | grep usbtest
```

Example output:

```
/dev/sdb1 on /mnt/usbtest type vfat
(rw,nosuid,nodev,relatime)
```

CLI example: unmount it safely

```
sudo umount /mnt/usbtest
```

Typical result:

```
(no output if successful)
```

Ubuntu: The Complete Guide explicitly discusses using `umount` toward a specific volume or mount point before removing or ejecting the drive.

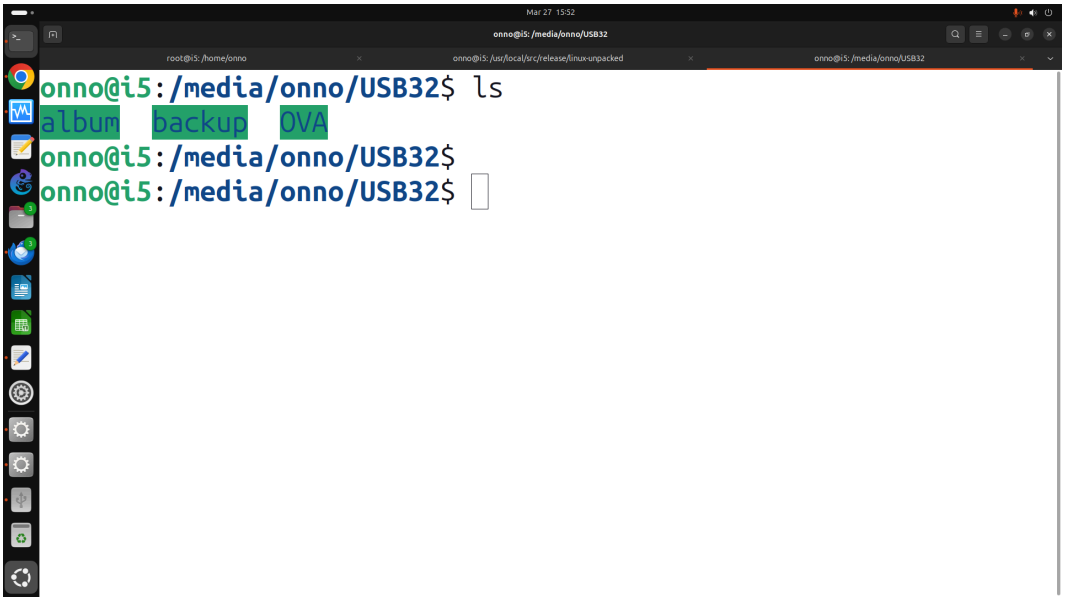


Figure. /media/student/USB32 to the visible USB drive.

Troubleshooting Peripherals and Device Detection

Peripheral troubleshooting begins with calm observation. The Ubuntu Linux Bible repeatedly emphasizes that when hardware is connected or disconnected, Linux generates messages that can be reviewed later with `dmesg`, `journalctl`, and system logs. It also explains that many device issues are easier to understand when users check whether the kernel detected the hardware and whether the relevant interface or mount point was created.

This leads to a useful troubleshooting sequence for printers, scanners, USB storage, and other peripherals:

- confirm the device is physically connected and powered
- check whether Ubuntu sees it with `lsusb`, `lspci`, or `lsblk`
- watch logs with `journalctl -f` or `tail -f /var/log/syslog`
- verify mount points or printer queue status
- try a different cable, port, or USB controller
- restart the relevant service if needed
- only then move on to driver or package troubleshooting

A good analogy is to think of peripheral troubleshooting like checking guests into the building. Before asking what job they should do, first confirm they actually entered the building and were registered correctly.

CLI example: inspect printer-related status

```
lpstat -t
```

Example output:

```
scheduler is running
system default destination: OfficePrinter
printer OfficePrinter is idle. enabled since Fri 14 Mar 2026
09:12:14 WIB
```

CLI example: inspect block devices

```
lsblk
```

Example output:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sdb      8:16   1  29.3G  0 disk
└─sdb1   8:17   1  29.3G  0 part /media/student/USB32
```

CLI example: inspect USB peripherals

```
lsusb
```

Example output:

```
Bus 001 Device 006: ID 03f0:3f17 HP, Inc. LaserJet Pro MFP M428fdw
Bus 001 Device 007: ID 0781:5583 SanDisk Corp. Ultra Fit
```

CLI example: inspect recent hardware messages

```
dmesg | tail -20
```

Example output:

```
[ 6051.113441] usb 1-3: new SuperSpeed USB device number 7 using xhci_hcd
[ 6051.249310] usb 1-3: New USB device found, idVendor=0781, idProduct=5583
[ 6051.250012] sd 6:0:0:0: [sdb] 62521344 512-byte logical blocks
[ 6051.330208]  sdb: sdb1
```

CLI example: monitor new events live

```
journalctl -f
```

Example output:

```
Mar 14 10:51:22 ubuntu-desktop udisksd[913]: Mounted
/dev/sdb1 at /media/student/USB32
Mar 14 10:51:30 ubuntu-desktop
system-config-printer[7143]: Added printer OfficePrinter
```

These commands are practical because they cover the most common peripheral categories in one troubleshooting flow.

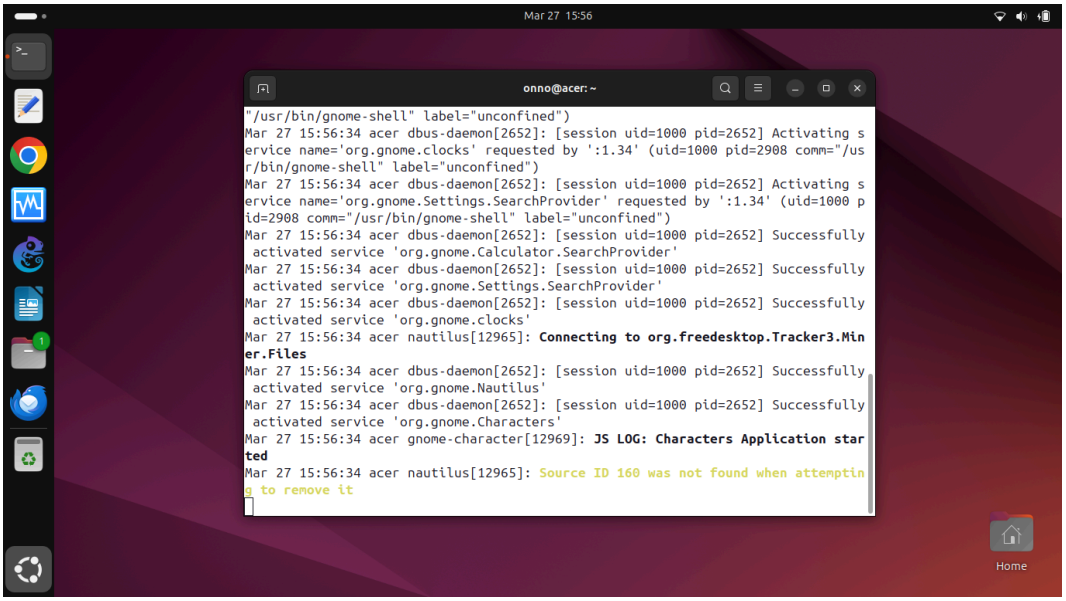


Figure. Terminal with journalctl -f open while a USB device is plugged in.

Guided Peripheral Management Lab

To make this chapter practical, readers can complete a short hands-on activity:

- install or confirm **CUPS** and **system-config-printer**
- add a test printer and run `lpstat -t`
- submit `/etc/hosts` as a print job with `lp`
- connect a multifunction device and identify it with `lsusb`
- follow connection events with `journalctl -f`
- insert a USB flash drive and inspect it with `lsblk -f`
- view mounted filesystems using `mount`
- manually mount a removable device at `/mnt/usbtest`
- safely unmount it with `umount`
- insert optical media and inspect whether Ubuntu mounted it automatically

This kind of lab turns peripheral handling from a vague desktop skill into a repeatable support practice.

Chapter Reflection

Printers, scanners, storage devices, and removable media are essential parts of daily Ubuntu Desktop use. Printing is built around **CUPS**, with graphical printer tools, queues, drivers, and network-printing support. Scanners and multifunction peripherals depend on correct detection and often appear as more than one device type. USB drives and other removable media are usually handled automatically through the kernel, **Udev**, and desktop automount behavior, while optical media may also be mounted and opened automatically. Understanding filesystems and mount points helps users see how Ubuntu connects physical devices to usable directories. When something goes wrong, tools such as `lsusb`, `lsblk`, `mount`, `lpstat`, `dmesg`, `journalctl`, and `tail -f /var/log/syslog` make troubleshooting much more systematic. Together, these skills help Ubuntu Desktop serve as a reliable workstation in homes, schools, labs, and offices.

Key Takeaways

- **CUPS is the core printing system on Ubuntu Desktop.**
- **The Printers window and CUPS tools** make adding and managing printers practical for desktop users.
- **Multifunction devices may appear as printer, scanner, and storage hardware at once.**
- **Ubuntu usually automounts supported removable media**, especially common USB filesystems.
- **Mount points are how filesystems become accessible in the desktop environment.**
- **Optical media can still be mounted and managed effectively on Ubuntu.**
- **Good peripheral troubleshooting starts with detection and logs, not guesswork.**

Part VII — Networking on Ubuntu Desktop

Chapter 15 — Network Configuration for Desktop Users

A modern Ubuntu desktop becomes truly useful only when it can **connect reliably, identify the right network, reach online services, and be diagnosed quickly when something goes wrong**. In the Ubuntu Academy outline, networking is one of the most important focus areas, covering **NetworkManager, Wi-Fi and Ethernet, static IP, DNS, proxy settings, VPN, mobile broadband, hostname basics, and quick troubleshooting with `ping`, `ip`, and `nmcli`**.

The attached references support this chapter from complementary angles. **Mastering Ubuntu** describes the beginner-friendly GNOME desktop workflow for connecting to Wi-Fi and explains that wired Ethernet usually connects automatically. **Ubuntu Linux Bible** goes deeper into practical network configuration, including static addressing, DNS resolution, route checking, and the role of NetworkManager. Together, they form a strong foundation for a desktop-focused networking chapter.

A useful illustration is to think of Ubuntu networking like the road system around a campus. The **network interface** is your vehicle, the **IP address** is your temporary parking spot or office number, the **router** is the gate to the outside world, **DNS** is the directory that translates names into actual destinations, and **NetworkManager** is the desk clerk that keeps all of those arrangements organized.

NetworkManager Basics

On Ubuntu Desktop, **NetworkManager** is the central tool for handling most everyday desktop networking tasks. The Ubuntu Linux Bible explicitly states that Ubuntu and most Linux systems using NetworkManager store wired and wireless configuration in connection files and that NetworkManager can automatically bring interfaces up using information obtained from DHCP. It also explains that NetworkManager's graphical interface can be used to configure static IP addresses, gateway systems, and DNS servers when needed.

For desktop users, this is extremely important because it means networking is not something they must build from scratch. In most cases, Ubuntu already knows how to:

- detect wired and wireless interfaces
- request an address from a DHCP server
- store connection profiles
- reconnect to known networks
- allow manual overrides when automatic settings are not enough

A helpful beginner model is this: **NetworkManager is the front desk of your desktop's network life**. It remembers trusted Wi-Fi networks, manages wired

connections, and lets the user switch between automatic and manual network settings without editing low-level files directly.

CLI example: list network devices with nmcli

```
nmcli device status
```

Example output:

```
DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     CampusNet
enp0s31f6   ethernet  disconnected  --
lo          loopback  unmanaged     --
```

This gives a fast desktop-friendly snapshot of what NetworkManager currently sees.

CLI example: show saved connection profiles

```
nmcli connection show
```

Example output:

```
NAME         UUID                                  TYPE
DEVICE
CampusNet    2d8a01a4-bd0b-4f0d-9c7f-6abf3be8f555  wifi
wlp2s0
Wired 1      f7b0cb11-48cb-4f1d-8ce8-798e84d92211  ethernet  --
```

This reinforces the idea that Ubuntu stores and reuses connection profiles instead of asking the user to rebuild the network every time.

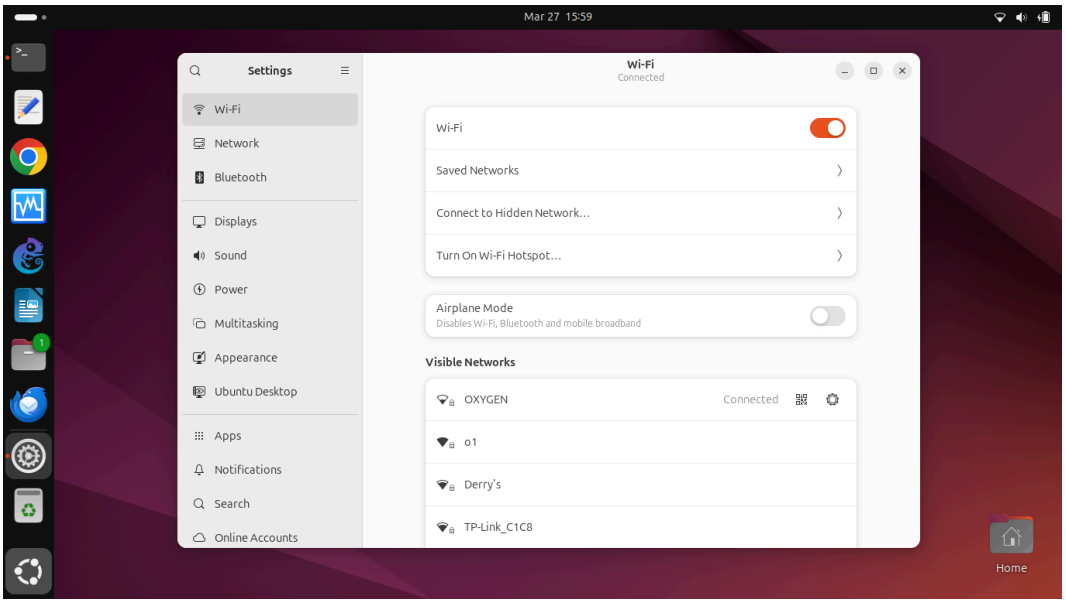


Figure. Quick Settings menu open, showing Wi-Fi configuration

Wi-Fi and Ethernet Setup

The beginner workflow for connecting Ubuntu Desktop to the internet is simple and well described in **Mastering Ubuntu**. It explains that users can press the **network icon on the top panel**, see a list of wireless networks, select one, and enter its key. It also notes that if the computer is connected by an Ethernet cable, Ubuntu will usually connect to the internet automatically.

That makes Ubuntu Desktop very approachable for normal users. The system behaves sensibly:

- **Wi-Fi** usually requires network selection and password entry
- **Ethernet** usually works automatically if the link is active and DHCP is available
- known wireless networks can be remembered for future automatic reconnection

A useful illustration is to imagine Wi-Fi as introducing yourself at a guarded entrance, while Ethernet is like entering through a pre-authorized office door.

CLI example: scan available Wi-Fi networks

```
nmcli device wifi list
```

Example output:

```
IN-USE  SSID         MODE   CHAN  RATE        SIGNAL  BARS
SECURITY
*       CampusNet    Infra  36    540 Mbit/s  78      ▂▄▆_  WPA2
        GuestNet     Infra  11    130 Mbit/s  52      ▂▄__  WPA2
        Lab-IoT      Infra  1     195 Mbit/s  34      ▂___  WPA2
```

CLI example: connect to a Wi-Fi network

```
nmcli device wifi connect "CampusNet" password
"example-password"
```

Example output:

```
Device 'wlp2s0' successfully activated with
'2d8a01a4-bd0b-4f0d-9c7f-6abf3be8f555'.
```

CLI example: inspect IP address state on Ethernet and Wi-Fi

```
ip a
```

Example output:

```
2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.1.24/24 brd 192.168.1.255 scope global dynamic wlp2s0
3: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500
```

This kind of output clearly shows that Wi-Fi is active and has an IPv4 address, while the Ethernet interface is present but not currently linked.

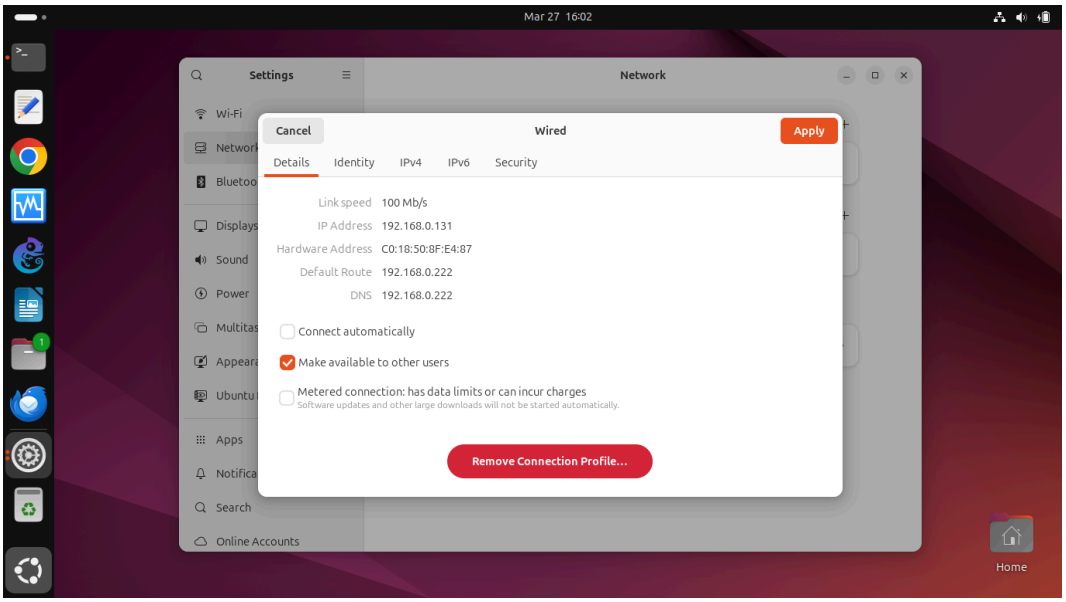


Figure. Ethernet settings page showing wired network con figuration.

IP Addressing Basics

IP addressing is one of the most important networking concepts for desktop users, but it does not need to be mysterious. The Ubuntu Linux Bible explains that when NetworkManager uses DHCP, the system automatically receives address information needed for startup. It also shows examples of route output where the local interface has an address such as `192.168.122.194` and where the default gateway points to the router that handles traffic beyond the local subnet.

A practical desktop explanation is:

- an **IP address** identifies your system on the current network
- the **subnet** describes the local network range
- the **default gateway** is the router used for traffic beyond the local network
- **DNS servers** translate names like `ubuntu.com` into addresses the system can reach

A helpful illustration is to imagine a university building. Your **IP address** is your room number, the **subnet** is your building wing, the **default gateway** is the main exit to the rest of campus, and **DNS** is the directory service that tells you where “Library” actually is.

CLI example: show addresses assigned to interfaces

```
ip a
```

Example output:

```
2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.1.24/24 brd 192.168.1.255 scope global dynamic wlp2s0
```

CLI example: show routing information

```
ip route show
```

Example output:

```
default via 192.168.1.1 dev wlp2s0 proto dhcp metric 600
192.168.1.0/24 dev wlp2s0 proto kernel scope link src 192.168.1.24 metric 600
```

This mirrors the kind of route explanation given in the Ubuntu Linux Bible, where the **default line** shows the gateway used to reach non-local destinations.

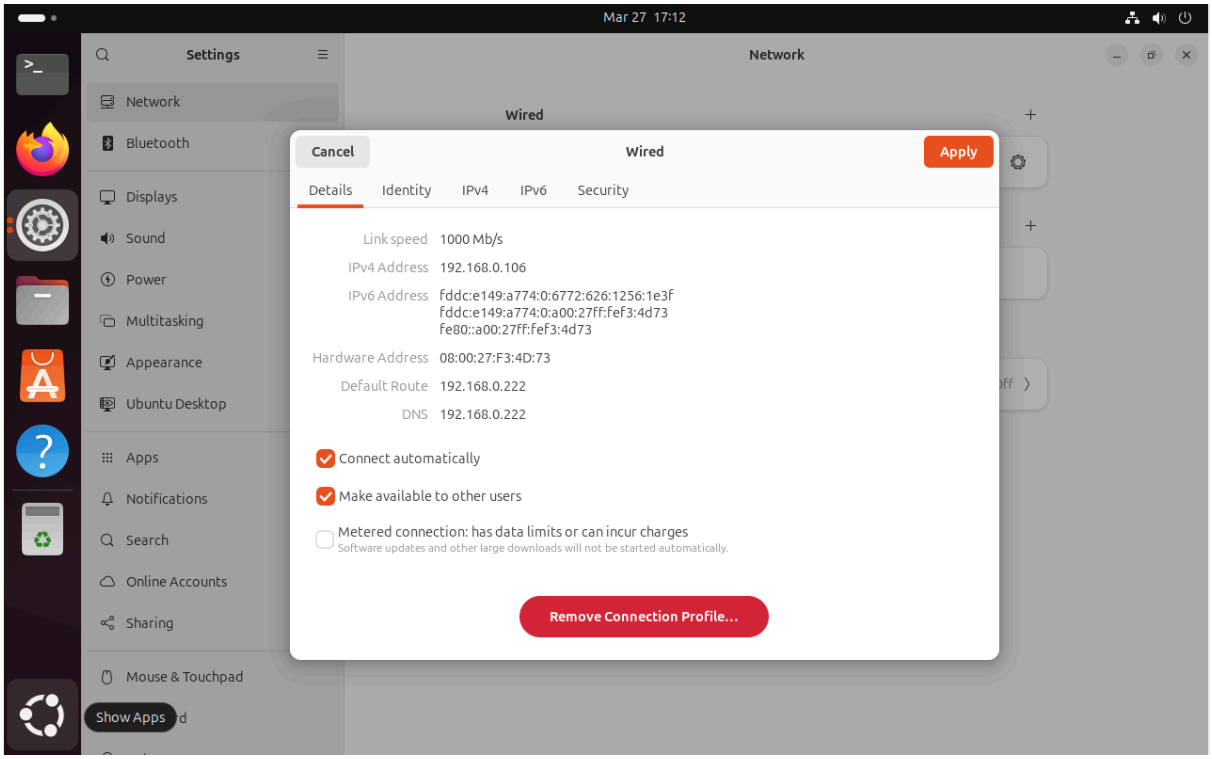


Figure. Settings > Network > active connection details showing IP address, gateway, and DNS entries

Static IP, DNS, and Proxy Configuration

The Ubuntu Linux Bible provides especially strong support for this section. It explains how to edit a connection, change IPv4 from **Automatic** to **Manual**, enter a specific IP address, define the gateway, specify one or two DNS servers, and optionally configure custom routes. It also notes that search domains are used when a host is requested without a fully qualified domain name.

This is the point where networking becomes more intentional. Many desktop users are happy with DHCP, but some environments require:

- a **static IP address** for printers, lab systems, kiosks, or administrative desktops
- manual **DNS servers** for internal name resolution or specific policies
- **proxy settings** in classrooms, offices, or controlled internet environments

The Ubuntu Linux Bible also discusses proxy servers in educational and business contexts, explaining that a proxy server can restrict network traffic and can be useful in places such as school computer labs.

A good illustration is to think of DHCP as being assigned a hotel room at check-in, while a static IP is like having a permanent office with your name on the door.

CLI example: set a static IPv4 address with nmcli

```
sudo nmcli connection modify "Wired 1" ipv4.method manual ipv4.addresses
192.168.1.50/24 ipv4.gateway 192.168.1.1 ipv4.dns "1.1.1.1 8.8.8.8"
sudo nmcli connection up "Wired 1"
```

Example output:

```
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/8)
```

CLI example: inspect current DNS configuration source

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

Example output:

```
search example.com
nameserver 1.1.1.1
nameserver 8.8.8.8
```

The Ubuntu Linux Bible explicitly explains that DNS server addresses ultimately appear in `/etc/resolv.conf` and that this file is part of the name-resolution path.

CLI example: verify DNS servers manually

```
host google.com 1.1.1.1
dig @8.8.8.8 google.com
```

Example output:

```
Using domain server:
Name: 1.1.1.1
Address: 1.1.1.1#53
Aliases:
google.com has address 142.250.190.14
google.com has IPv6 address 2404:6800:4003:c00::66
```

The Ubuntu Linux Bible specifically recommends `host` or `dig` against a chosen DNS server to verify whether that server can resolve a hostname properly.

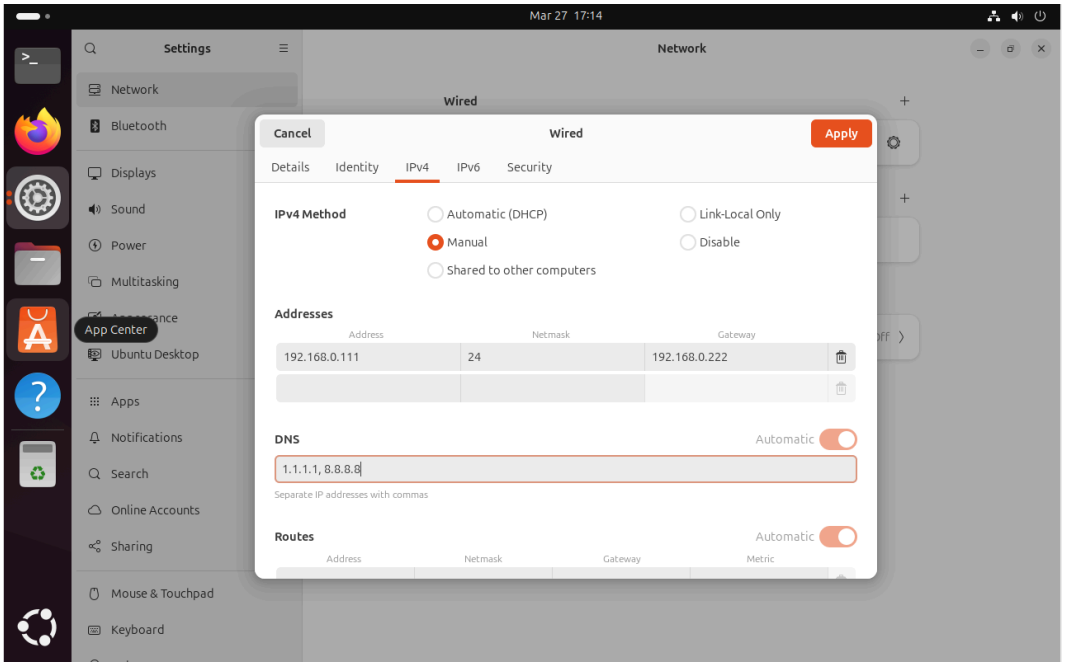


Figure.IPv4 configuration screen in Network settings with Manual selected and static IP fields filled

VPN and Mobile Broadband

The Ubuntu Academy outline explicitly includes **VPN and mobile broadband configuration** as part of desktop networking capability. That means this book should treat them as real desktop tasks, not advanced server topics.

Within the attached references, the strongest recurring theme is that Ubuntu Desktop networking is managed through **NetworkManager**, which means VPN and mobile broadband profiles fit naturally into the same desktop connection system used for Wi-Fi and Ethernet. The same connection-profile logic applies: the system stores settings, activates connections, and can switch between them through the desktop UI or `nmcli`.

A practical desktop explanation is:

- a **VPN** extends a secure private path through another network
- **mobile broadband** uses a cellular modem or tethered device to provide internet access
- both are usually managed as **connection profiles**, just like Wi-Fi and Ethernet, rather than as completely separate networking worlds

A helpful illustration is to imagine VPN as a private tunnel through a public road system, while mobile broadband is like using a different transportation network entirely when Wi-Fi or wired access is unavailable.

CLI example: list available NetworkManager connection types

```
nmcli connection show
```

Example output:

```
NAME          UUID                                  TYPE      DEVICE
CampusNet     2d8a01a4-bd0b-4f0d-9c7f-6abf3be8f555  wifi      wlp2s0
Work-VPN      7b315e72-21bc-4e03-a111-9a8c4a52ce55 vpn       --
Wired 1       f7b0cb11-48cb-4f1d-8ce8-798e84d92211 ethernet  --
```

CLI example: bring up a saved VPN profile

```
nmcli connection up "Work-VPN"
```

Example output:

```
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/12)
```

CLI example: display current device status after VPN activation

```
nmcli device status
```

Example output:

```
DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     CampusNet
tun0        tun       connected     Work-VPN
lo          loopback  unmanaged     --
```

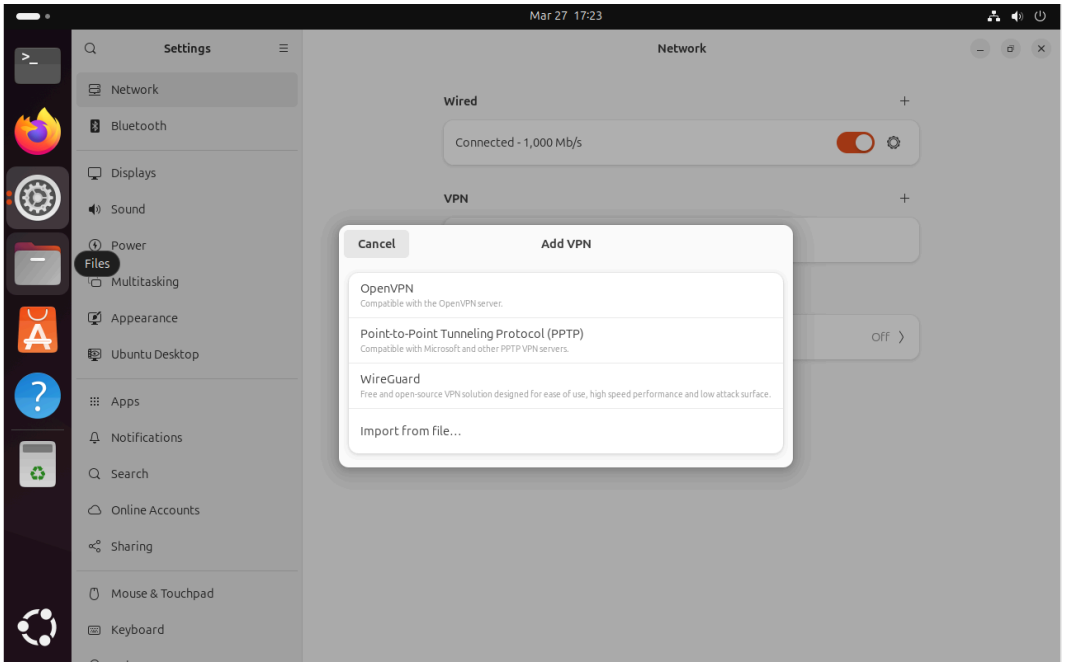


Figure. Settings > VPN showing an added VPN profile

Hostname Configuration

Every Ubuntu system has a **hostname**, which is the local name the system uses to identify itself on a network and to the user. The Ubuntu Academy outline explicitly includes **basic hostname and domain setup**, which makes hostname awareness part of desktop administration rather than just server administration.

Hostname matters in practical desktop life because it appears in:

- terminal prompts
- support notes and documentation
- local network identification
- file-sharing or remote-access contexts
- multi-machine lab environments where systems need clear names

A good illustration is to think of the hostname as the nameplate on an office door. The building may still function without a memorable label, but support and collaboration become much easier when every room is clearly identified.

CLI example: show the current hostname

```
hostname
hostnamectl
```

Example output:

```
ubuntu-desktop
 Static hostname: ubuntu-desktop
       Icon name: computer-laptop
         Chassis: laptop
Operating System: Ubuntu 24.04 LTS
          Kernel: Linux 6.8.0-52-generic
    Architecture: x86-64
```

CLI example: set a new hostname

```
sudo hostnamectl set-hostname lab-pc-01
hostnamectl
```

Example output:

```
 Static hostname: lab-pc-01
       Icon name: computer-laptop
         Chassis: laptop
```

This is an especially useful habit in classrooms, labs, and shared environments where “ubuntu-desktop” is too generic to be useful.

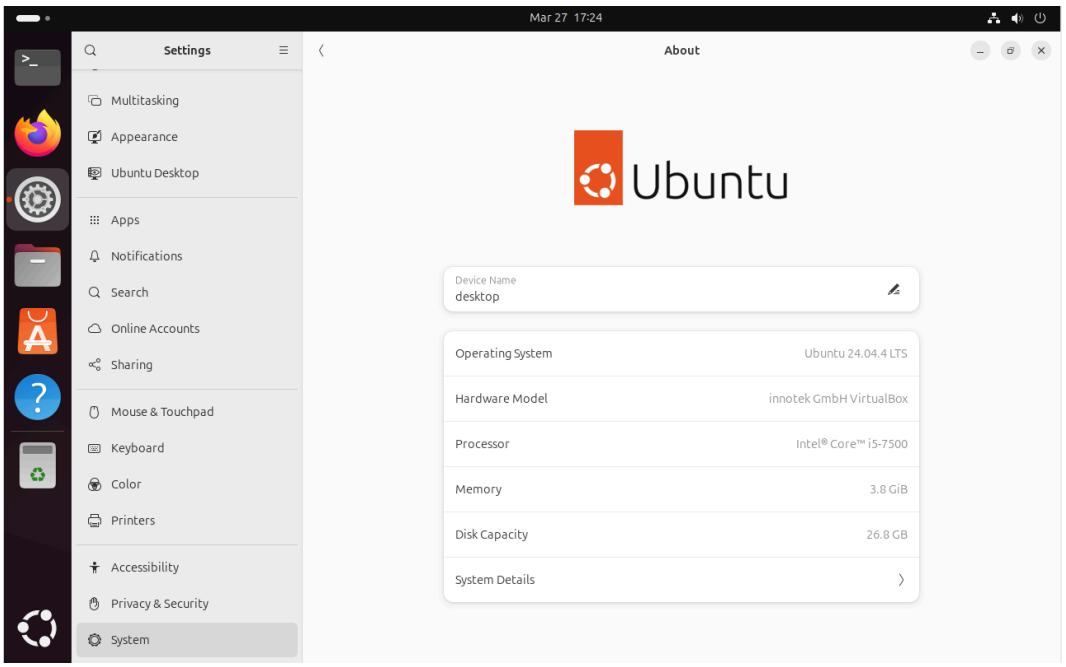


Figure. Settings > About showing the device name / hostname

Connection Profiles and Troubleshooting

The Ubuntu Linux Bible makes an important point: with NetworkManager, wired and wireless interfaces are stored as **connection profiles**, and those profiles can be edited to change addressing, routing, or name-resolution behavior. It also explains that debugging often begins by checking routes, gateway reachability, and whether the system can reach a public IP address or only fails at name resolution.

This gives desktop users a smart troubleshooting sequence:

- is the **interface up**?
- does it have an **IP address**?
- is the **default route** present?
- can the system reach the **gateway**?
- can it reach a **public IP address**, such as `8.8.8.8`?
- can it resolve **hostnames** correctly?

A useful analogy is to think of troubleshooting like checking a travel route. First confirm you can leave your room, then your building, then campus, then the city, and finally verify that the map can find named destinations.

CLI example: inspect a specific connection profile

```
nmcli connection show "CampusNet"
```

Example output:

```
connection.id:    CampusNet
connection.type:  802-11-wireless
ipv4.method:      auto
ipv4.dns:         --
GENERAL.STATE:    activated
```

CLI example: reconnect a profile

```
nmcli connection down "CampusNet"
nmcli connection up "CampusNet"
```

Example output:

```
Connection 'CampusNet' successfully deactivated
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/14)
```

CLI example: check route to the gateway and beyond

```
ip route show
ping -c 2 192.168.1.1
ping -c 2 8.8.8.8
```

Example output:

```
default via 192.168.1.1 dev wlp2s0 proto dhcp metric 600
192.168.1.0/24 dev wlp2s0 proto kernel scope link src 192.168.1.24
metric 600

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
```

```
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.74 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.69 ms

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=22.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=22.8 ms
```

The Ubuntu Linux Bible explicitly recommends this style of troubleshooting: check the default route, ping the router, then ping a well-known public IP to separate routing problems from DNS problems.

CLI example: test hostname resolution separately

```
ping -c 2 archive.ubuntu.com
```

Example output:

```
PING archive.ubuntu.com (185.125.190.81) 56(84) bytes of data.
64 bytes from archive.ubuntu.com: icmp_seq=1 ttl=52 time=188 ms
64 bytes from archive.ubuntu.com: icmp_seq=2 ttl=52 time=189 ms
```

If public IPs work but named hosts do not, the references indicate the next place to look is DNS resolution and /etc/resolv.conf.

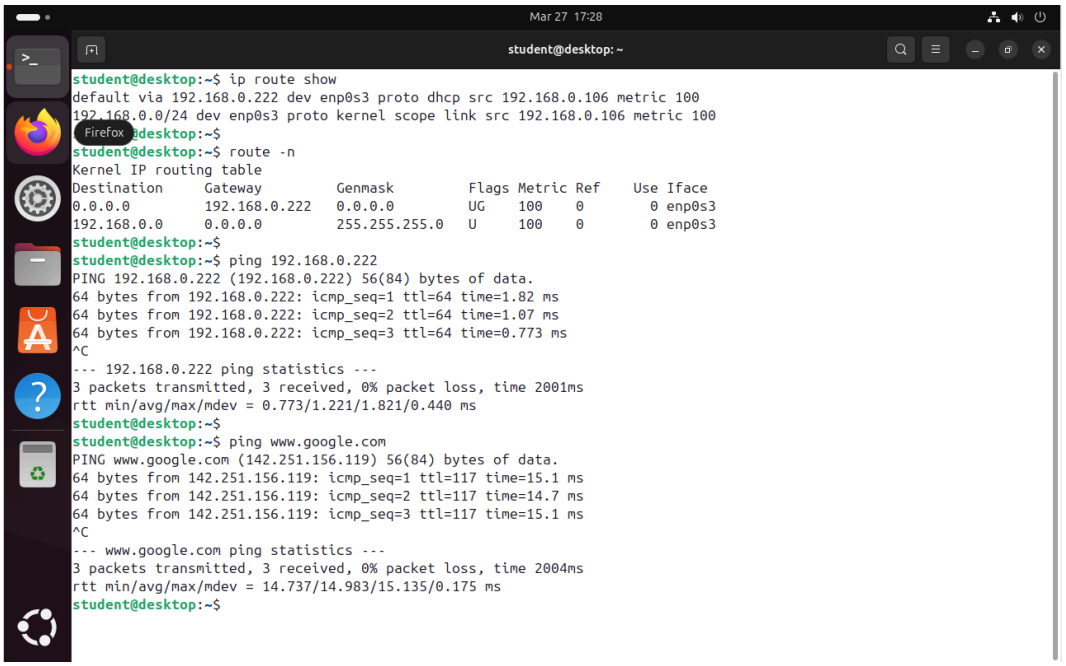


Figure. Terminal showing troubleshooting flow: ip route, ping gateway, ping public IP, ping hostname

Using `ping`, `ip`, and `nmcli` for Quick Diagnostics

The Ubuntu Academy outline explicitly names `ping`, `ip`, and `nmcli` as the key quick-diagnostic tools for desktop networking. That is a very good choice because these three commands cover the most common support questions without overwhelming the learner.

Each one answers a different class of question:

- `ping` → Can I reach this host?
- `ip` → What addresses and routes do I currently have?
- `nmcli` → What does NetworkManager think is happening?

A practical desktop support habit is to use them together in this order:

1. **`nmcli device status`** — is the interface connected?
2. **`ip a`** — does it have an address?
3. **`ip route show`** — does it have a route out?
4. **`ping` the gateway** — can it reach the router?
5. **`ping` a public IP** — can it reach the internet?
6. **`ping` a hostname** — is DNS working too?

This is a powerful diagnostic ladder because it narrows the problem stage by stage, exactly in the spirit of the Ubuntu Linux Bible's route and DNS troubleshooting guidance.

CLI example: fast all-in-one network check

```
nmcli device status
ip a
ip route show
ping -c 2 192.168.1.1
ping -c 2 8.8.8.8
ping -c 2 archive.ubuntu.com
```

Example output:

```
DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     CampusNet

2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.1.24/24 brd 192.168.1.255 scope global dynamic wlp2s0

default via 192.168.1.1 dev wlp2s0 proto dhcp metric 600
192.168.1.0/24 dev wlp2s0 proto kernel scope link src 192.168.1.24 metric 600

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.73 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.70 ms

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=22.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=22.7 ms
```

```
PING archive.ubuntu.com (185.125.190.81) 56(84) bytes of data.
64 bytes from archive.ubuntu.com: icmp_seq=1 ttl=52 time=188 ms
64 bytes from archive.ubuntu.com: icmp_seq=2 ttl=52 time=189 ms
```

This is exactly the kind of practical command sequence that turns a desktop user into a capable desktop troubleshooter.

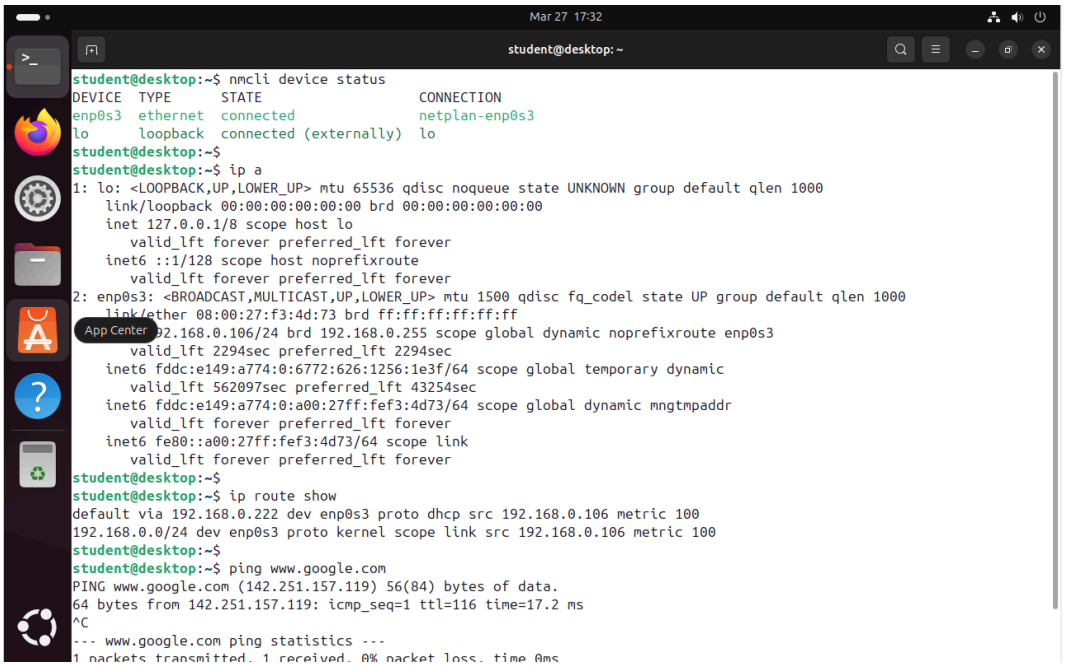


Figure. Terminal showing nmcli, ip, and ping used together for quick troubleshooting.

Guided Networking Lab

To make this chapter practical, readers can complete a short network-administration exercise:

- connect to a Wi-Fi network through the desktop UI
- confirm the connection with `nmcli device status`
- inspect the assigned IP with `ip a`
- inspect the default route with `ip route show`
- ping the local router and then `8.8.8.8`
- ping a hostname such as `archive.ubuntu.com`
- create or inspect a saved connection profile with `nmcli connection show`
- modify a test connection to use a **static IP**
- verify DNS using `host` or `dig`
- change the hostname of the desktop and confirm it with `hostnamectl`

This kind of exercise helps readers see networking not as abstract theory, but as a set of practical desktop skills they can repeat and support.

Chapter Reflection

Network configuration on Ubuntu Desktop is built around clarity, automation, and practical control. **NetworkManager** handles most everyday desktop connectivity, including Wi-Fi, Ethernet, VPN, and other profiles, while still allowing manual configuration when DHCP is not enough. Wi-Fi and Ethernet setup are straightforward for normal users, but deeper understanding comes from learning how IP addressing, routes, gateways, and DNS fit together. Static IP configuration, custom DNS, and proxy settings are especially important in classrooms, labs, and office environments. Hostnames help make systems identifiable and manageable. Most importantly, quick diagnostic tools such as `ping`, `ip`, and `nmcli` let desktop users troubleshoot methodically instead of guessing. By combining graphical settings with a few reliable command-line habits, Ubuntu Desktop users gain a networking skill set that is both practical and professional.

Key Takeaways

- **NetworkManager is the main networking framework for Ubuntu Desktop users.**
- **Wi-Fi is usually chosen from the desktop menu, while Ethernet often connects automatically.**
- **IP addressing, gateways, and DNS** are the core concepts behind successful desktop networking.
- **Static IP, DNS, and proxy settings** matter most in managed, educational, or office networks.
- **VPN and mobile broadband fit naturally into the same connection-profile model used by NetworkManager.**
- **Hostnames make desktop systems easier to identify and support.**
- **`ping`, `ip`, and `nmcli` form an excellent quick-diagnostics toolkit.**

Chapter 16 — Desktop Sharing and Connectivity

A desktop is not only a personal workspace. It is also a **connection point**. Ubuntu Desktop becomes much more useful when it can exchange files with nearby devices, open shared folders across a network, allow remote access when needed, and adapt to different environments such as home, campus, and office networks. In the Ubuntu Academy outline, this chapter fits naturally under **networking capabilities** and practical desktop administration, especially around **sharing internet, troubleshooting connectivity, and everyday desktop use**.

The attached references support this chapter in several ways. The Ubuntu Academy outline explicitly calls out **sharing internet (hotspot)** and broader desktop networking tasks. *Ubuntu: The Complete Guide* gives concrete guidance for **Samba/SMB folder access**, **proxy-aware network use**, and connection workflows on home and workplace networks. *Ubuntu Linux Bible* adds background on **Samba daemons**, **SMB access with smbclient**, and how Linux desktop systems usually rely on **NetworkManager** for day-to-day connectivity.

A useful illustration is to think of Ubuntu Desktop as a well-connected campus office. **Bluetooth sharing** is like handing a file to someone standing next to you. **Network file sharing** is like accessing another room in the same building. **Remote desktop** is like controlling your office from another location. **Secure transfer tools** are like locked delivery envelopes. **Hotspot sharing** is like turning your office into a temporary internet gateway for others.

Bluetooth Sharing and Local Transfers

Bluetooth is one of the simplest ways to connect nearby devices without needing a cable or a full network share. In the broader Ubuntu Desktop context, Bluetooth support appears alongside sound, pairing, and device management in the Ubuntu Academy outline, which makes it a normal part of desktop connectivity rather than a niche feature.

For desktop users, Bluetooth sharing usually involves:

- pairing a phone, headset, keyboard, mouse, or speaker
- sending or receiving small files locally
- using nearby-device transfers when Wi-Fi sharing is unnecessary
- treating Bluetooth as a **short-range convenience channel**, not a heavy file-server replacement

A practical way to explain Bluetooth sharing is to compare it to passing a USB stick across a desk, except the “stick” is invisible and the devices must recognize each other first.

Before local Bluetooth transfers can work smoothly, the system usually needs:

- a visible and working Bluetooth adapter
- the target device in pairing mode
- acceptance of the pairing request
- in some cases, approval for file transfer or sharing access

CLI example: verify that Ubuntu sees a Bluetooth adapter

```
lsusb | grep -i bluetooth
```

Example output:

```
Bus 001 Device 002: ID 8087:0a2b Intel Corp. Bluetooth
wireless interface
```

CLI example: check Bluetooth service status

```
systemctl status bluetooth
```

Example output:

```
● bluetooth.service - Bluetooth service
     Loaded: loaded (/lib/systemd/system/bluetooth.service; enabled)
     Active: active (running) since Fri 2026-03-14 08:12:04 WIB
```

CLI example: open the Bluetooth settings page

```
gnome-control-center bluetooth &
```

Example output:

```
[1] 7032
```

These commands help bridge graphical pairing with system-level verification, which is very useful in classrooms or support scenarios where a device “does not show up” at first. The Ubuntu Academy outline’s Bluetooth focus supports exactly this kind of practical workflow.

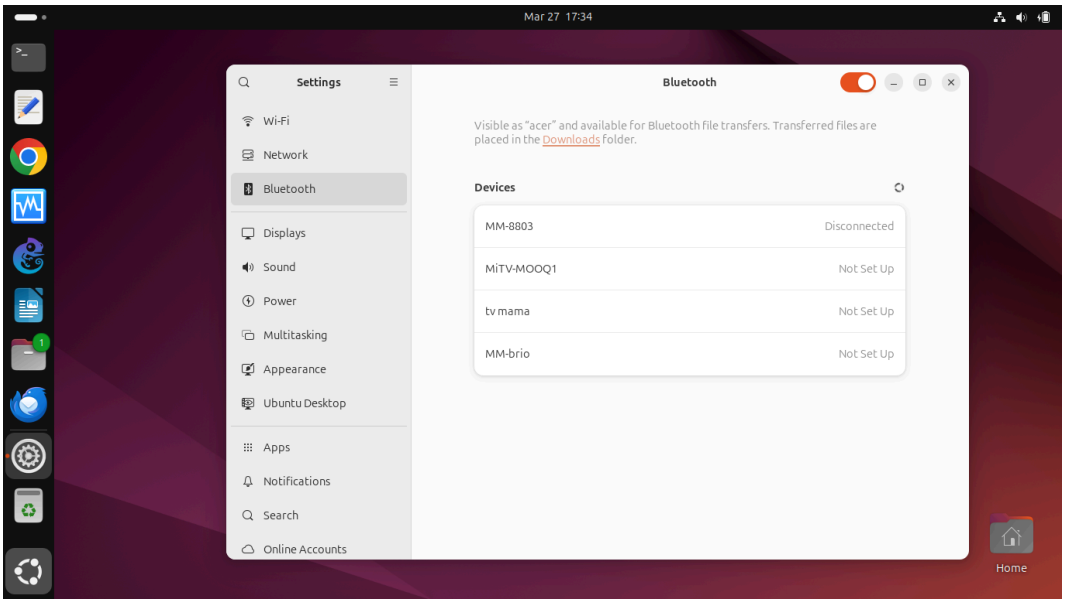


Figure. Settings > Bluetooth showing Bluetooth enabled and a nearby device ready to pair

Network File Access Basics

One of the most useful everyday desktop skills is opening files stored on another machine. *Ubuntu: The Complete Guide* gives especially strong support here. It explains that if users need to access folders on other machines in the home or workplace, **Samba or SMB** is the common protocol, and that on Ubuntu the normal workflow is to open **Files**, choose **Other Locations**, and connect to an address such as `smb://192.168.42.1`. It also notes that once the folder is accessible, the user can **Add Bookmark** to make it easy to reach later.

This is a very important desktop lesson because it means Ubuntu can work naturally in mixed environments:

- Ubuntu ↔ Ubuntu
- Ubuntu ↔ Windows
- Ubuntu ↔ macOS

In real life, this matters in classrooms, offices, and homes where not every machine runs the same operating system.

A practical desktop model is:

- **local folders** live on your own system
- **network folders** live on another device but appear through Files as if they were another location
- **bookmarks** make repeated access easier
- permissions on the remote system still control what you can actually read or write

A good illustration is to imagine the Files application as a building directory. Your own folders are rooms in your office. SMB folders are rooms in another office down the hall, connected by a network corridor.

CLI example: open a Samba location from the command line

```
xdg-open smb://192.168.42.1/shared
```

Typical result:

```
(opens the Files application at the remote SMB location)
```

CLI example: check whether Samba client tools are available

```
which smbclient
```

Example output:

```
/usr/bin/smbclient
```

CLI example: list shares available on a Samba server

```
smbclient -L //192.168.42.1 -U student
```

Example output:

```
Enter WORKGROUP\student's password:

        Sharename       Type      Comment
        ---------       ----      -------
        documents       Disk      Shared Documents
        print$          Disk      Printer Drivers
        IPC$            IPC       IPC Service
```

The Ubuntu Linux Bible explicitly shows smbclient -L localhost -U chris as a way to verify that Samba shares exist and are reachable, which makes it a good CLI companion to the Files-based SMB workflow.

CLI example: connect to a share interactively

```
smbclient -U student //192.168.42.1/documents
```

Example output:

```
Enter WORKGROUP\student's password:
Try "help" to get a list of possible commands.
smb: \> ls
  .                   D        0  Fri Mar 14 09:10:00 2026
  assignments         D        0  Fri Mar 14 09:09:00 2026
  notes.txt           A     2048  Fri Mar 14 09:08:00 2026
```

This demonstrates the same underlying SMB access concept that the graphical Files application hides for convenience.

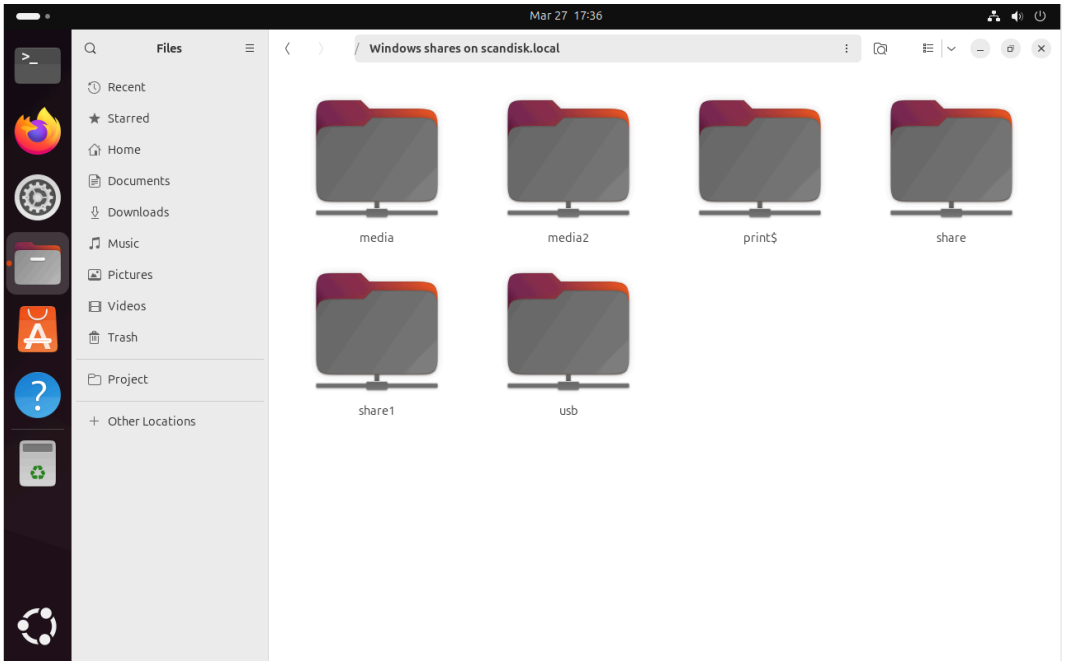


Figure. Files > Other Locations showing an SMB share being accessed

Remote Desktop and Screen Sharing

Remote access is one of the most practical Ubuntu Desktop features for support, teaching, demonstrations, and hybrid work. The Ubuntu Academy outline already points users toward desktop-oriented networking and sharing scenarios, and the modern Ubuntu desktop stack described in the references emphasizes improved screen-sharing capability in GNOME-based environments.

A desktop user should understand the difference between:

- **screen sharing** — letting another person view or help with the current desktop session
- **remote desktop** — controlling a desktop session from another machine
- **file sharing** — exchanging files without directly sharing the screen

These are different tools for different goals.

A good illustration is to imagine screen sharing as opening your classroom projector feed to someone else, while remote desktop is handing them the keyboard and mouse from afar.

In Ubuntu Desktop contexts, remote sharing is most useful for:

- helping a student or colleague solve a problem
- presenting a workflow on another machine
- connecting to a lab PC from another room
- allowing limited remote access to a desktop when physical access is inconvenient

Because the attached Ubuntu Academy outline also references firewall rules “for SSH/VNC” under networking capabilities, it clearly expects desktop users to understand that remote access often depends on both the desktop feature and the network policy permitting it.

CLI example: identify the current hostname before sharing remotely

```
hostnamectl
```

Example output:

```
Static hostname: lab-pc-01
      Icon name: computer-desktop
        Chassis: desktop
Operating System: Ubuntu 24.04 LTS
```

A clear hostname makes remote support much easier, especially in labs and shared environments.

CLI example: open the sharing-related desktop settings

```
gnome-control-center sharing &
```

Example output:

```
[1] 7196
```

CLI example: verify local address before remote access

```
ip a
```

Example output:

```
2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.1.24/24 brd 192.168.1.255 scope global dynamic wlp2s0
```

This helps the user know which address another device on the same LAN should target.

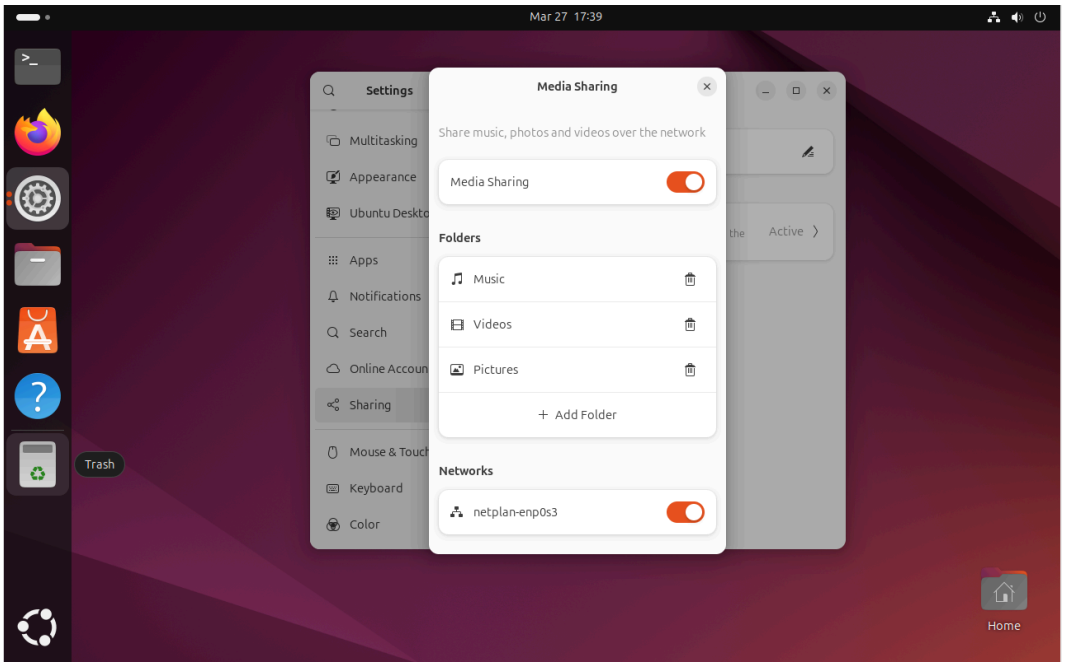


Figure. Settings > Sharing showing Media Sharing.

Secure File Transfer Basics

Not every file exchange should happen through open shares or informal drag-and-drop methods. In many cases, users need a more controlled and secure transfer path. The Ubuntu Academy outline does not name SSH explicitly in the networking section, but it does reference firewall rules for SSH and practical desktop networking support. That makes **secure transfer basics** a natural fit here.

A good desktop-level secure-transfer toolkit usually includes:

- **SCP** for quick secure file copying between two systems
- **SFTP** for secure file browsing and transfer using SSH
- SMB only when the sharing model and permissions are appropriate
- avoiding casual transfer over untrusted networks without encryption

A useful analogy is this: SMB sharing is like giving someone access to a filing cabinet in the same office. **SCP/SFTP** is like sending files through a locked courier envelope.

CLI example: copy a file securely with scp

```
scp ~/Documents/report.pdf student@192.168.1.50:/home/student/
```

Example output:

```
student@192.168.1.50's password:
report.pdf       100%  812KB   5.4MB/s   00:00
```

CLI example: copy a directory recursively

```
scp -r ~/Documents/Ubuntu-Academy
student@192.168.1.50:/home/student/
```

Example output:

```
student@192.168.1.50's password:
chapter1.md      100%   14KB   1.2MB/s   00:00
chapter2.md      100%   18KB   1.3MB/s   00:00
```

CLI example: open an SFTP session

```
sftp student@192.168.1.50
```

Example output:

```
student@192.168.1.50's password:
Connected to 192.168.1.50.
sftp>
```

These are simple but powerful workflows for secure transfers on local or managed networks.

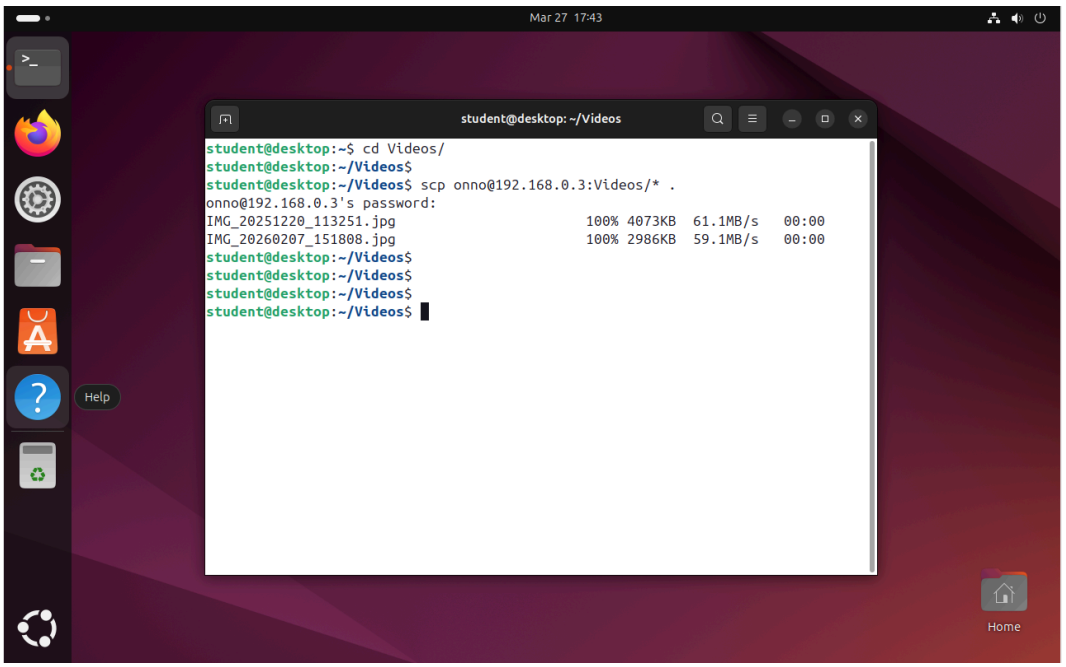


Figure. Terminal showing an scp file transfer completing successfully

Internet Sharing / Hotspot Setup

The Ubuntu Academy outline explicitly includes **sharing internet (hotspot)** as part of desktop networking capabilities. That makes hotspot setup a required desktop skill, not just an optional trick.

Hotspot or internet sharing is useful when:

- a laptop has internet over Ethernet and needs to share it over Wi-Fi
- a desktop needs to provide temporary local access in a classroom or lab
- mobile broadband is connected and needs to be shared with nearby devices
- a quick local network is needed without relying on existing infrastructure

A helpful illustration is to think of hotspot mode as turning your Ubuntu laptop into a small temporary router. Instead of only using the internet for itself, the system becomes a gateway for other devices.

Because the reference material treats proxy servers and managed access as realistic lab and workplace tools, it also reminds us that shared internet access should be treated responsibly. A hotspot is convenient, but it is still part of a larger network and security environment.

CLI example: review Wi-Fi device capability and state

```
nmcli device status
```

Example output:

```
DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     CampusNet
enp0s31f6   ethernet  connected     Wired 1
lo          loopback  unmanaged     --
```

CLI example: create a Wi-Fi hotspot with NetworkManager

```
nmcli device wifi hotspot ifname wlp2s0 ssid
UbuntuHotspot password "Ubuntu12345"
```

Example output:

```
Device 'wlp2s0' successfully activated with
'e2c0d4d5-3b97-43fb-bf11-3b6b5c8d8e0a'.
Hint: "nmcli connection show" to see active connection.
```

CLI example: inspect the created hotspot connection

```
nmcli connection show
```

Example output:

```
NAME    UUID                                  TYPE     DEVICE
Hotspot e2c0d4d5-3b97-43fb-bf11-3b6b5c8d8e0a wifi     wlp2s0
Wired 1 f7b0cb11-48cb-4f1d-8ce8-798e84d92211 ethernet enp0s31f6
```

This is one of the most practical desktop networking tasks because it shows how Ubuntu can shift from being just a client to temporarily acting as a service point for others.

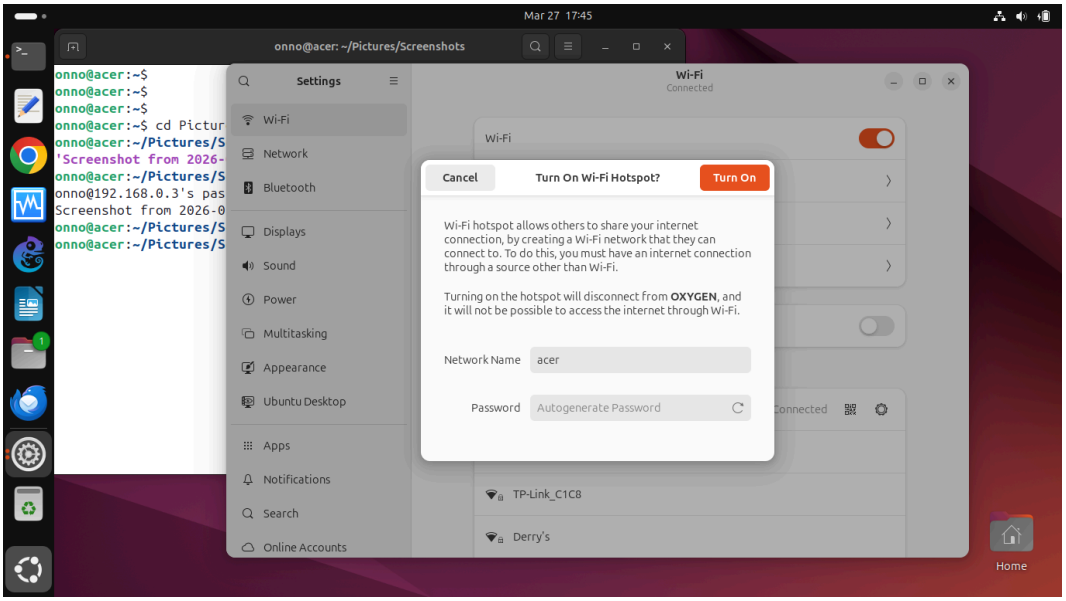


Figure. Ubuntu network settings showing a hotspot or shared connection active

Working on Home, Campus, and Office Networks

Not all networks behave the same way. *Ubuntu: The Complete Guide* is especially helpful here because it explicitly frames Ubuntu networking in **home** and **workplace** contexts, including proxy configuration, Wi-Fi selection, and access to shared folders on other devices. It explains that if a home or workplace uses a **proxy server**, users can configure that in the Network Manager or network settings area, and it also notes that users should verify whether they are connected to the correct wireless network rather than accidentally using an unsecured hotspot.

That makes this an important desktop lesson: network behavior depends heavily on the environment.

A practical comparison:

- **Home network**
  - usually simpler
  - often automatic DHCP
  - may involve local file sharing, printers, and media devices
- **Campus network**
  - may require more careful Wi-Fi selection
  - may have captive portals, access controls, or student/staff SSIDs
  - may rely on shared resources and lab machines
- **Office network**
  - may use proxy servers, managed shares, stricter policies, VPN requirements, and named desktops

A useful analogy is to think of each environment as a different building with different entry rules. At home, the door may always be open to known devices. On campus, you may need to sign in at the gate. In the office, there may be reception, ID badges, and approved corridors only.

CLI example: inspect connection information quickly

```
nmcli device show wlp2s0
```

Example output:

```
GENERAL.DEVICE:         wlp2s0
GENERAL.TYPE:           wifi
GENERAL.CONNECTION:     CampusNet
IP4.ADDRESS[1]:         192.168.1.24/24
IP4.GATEWAY:            192.168.1.1
IP4.DNS[1]:             1.1.1.1
```

CLI example: show current proxy-related environment variables

```
env | grep -i proxy
```

Example output:

```
http_proxy=http://proxy.example.org:3128
https_proxy=http://proxy.example.org:3128
```

This is useful in campus or office environments where proxy settings affect browser behavior, package downloads, or command-line tools.

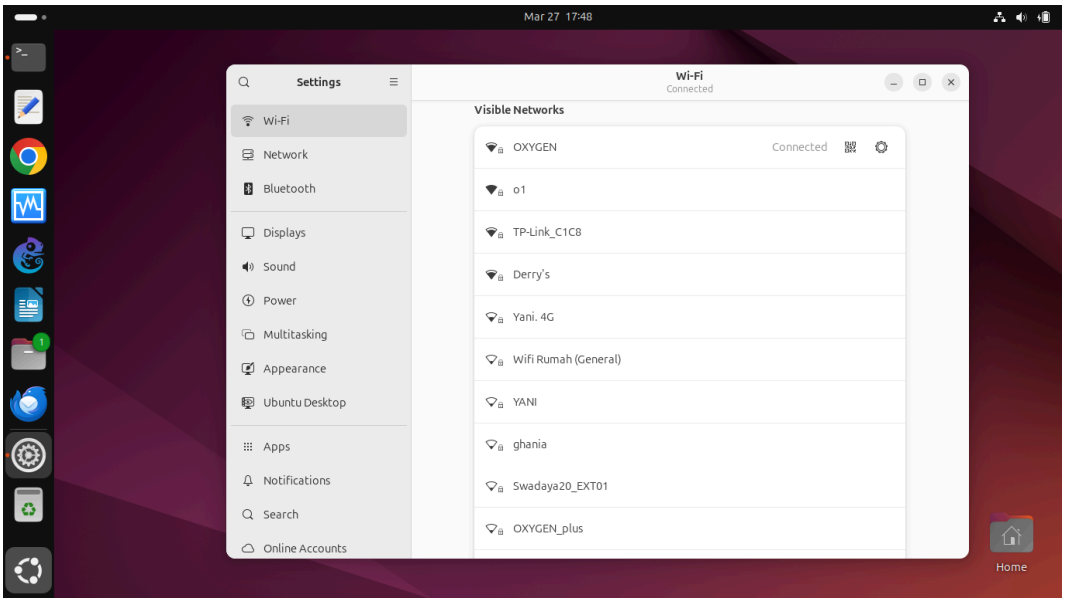


Figure. Wi-Fi list showing multiple networks, with the correct campus or office SSID selected

Practical Connectivity Scenarios for Daily Desktop Use

The real value of desktop connectivity is not in memorizing tools. It is in solving ordinary work situations calmly and efficiently. The Ubuntu Academy outline emphasizes hands-on desktop administration, and the attached references reinforce exactly that spirit: practical connectivity, easy desktop access to shared folders, remembered connections, and fast troubleshooting.

A few realistic scenarios help tie the chapter together.

Scenario 1 — Connect to campus Wi-Fi and verify access

- join the SSID from the top-right menu
- confirm with nmcli device status
- test with a browser or ping

CLI example

```
nmcli device status
ping -c 2 8.8.8.8
```

Example output:

```
DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     CampusNet

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=22.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=22.7 ms
```

Scenario 2 — Open a shared folder from a Windows or Mac machine

- open Files
- choose Other Locations
- enter smb://...
- bookmark the folder if it will be used regularly

Scenario 3 — Send a file securely to a colleague on the same LAN

- use scp
- verify the file arrived
- avoid casual insecure transfer paths on sensitive networks

CLI example

```
scp ~/Documents/notes.txt student@192.168.1.50:/home/student/
```

Scenario 4 — Turn a laptop into a temporary hotspot

- create the hotspot
- connect a phone or second laptop
- disable it when the temporary need ends

Scenario 5 — Diagnose “internet works by IP but not by name”

- ping the router
- ping a public IP
- then test DNS resolution, just as the networking references recommend

A useful final illustration is to imagine daily connectivity as a toolkit with several drawers. One drawer is for nearby-device sharing, one for shared folders, one for secure transfers, one for remote access, and one for temporary internet sharing. Ubuntu Desktop becomes powerful when users know which drawer to open for which task.

Guided Sharing and Connectivity Lab

To make this chapter practical, readers can complete a short hands-on exercise:

- pair a Bluetooth device and verify the Bluetooth service
- open a remote SMB share through **Files > Other Locations**
- bookmark the network folder for future access
- run `smbclient -L` against a test server
- transfer a file with `scp`
- inspect the current hostname with `hostnamectl`
- create a Wi-Fi hotspot with `nmcli device wifi hotspot`
- confirm the hotspot appears as a saved connection
- compare how Ubuntu behaves on a home-style Wi-Fi, a managed campus network, and a proxy-based office network

This kind of exercise helps learners move from abstract connectivity terms to practical desktop confidence.

Chapter Reflection

Desktop sharing and connectivity turn Ubuntu from a standalone workstation into a collaborative platform. Bluetooth supports nearby-device convenience and local transfers. Samba/SMB-based network access allows Ubuntu to work naturally with shared folders on Ubuntu, Windows, and macOS systems, especially through the Files application and saved bookmarks. Remote desktop and screen sharing support teaching, support, and flexible access. Secure transfer tools such as SCP and SFTP provide safer alternatives when files need to move across the network with more control. Hotspot mode lets Ubuntu temporarily share internet access with nearby devices. Finally, the way these tools are used depends on the environment: home, campus, and office networks all shape how desktop connectivity should be configured and trusted. Together, these capabilities make Ubuntu Desktop not just connected, but **usefully connected**.

Key Takeaways

- **Bluetooth sharing is best for nearby-device convenience and small local transfers.**
- **SMB/Samba is the most practical cross-platform file-sharing method for Ubuntu Desktop users.**
- **Files > Other Locations** is a core graphical path for network folder access.
- **Remote desktop and screen sharing** are different from file sharing and should be understood separately.
- **SCP and SFTP** are strong secure-transfer basics for desktop users.
- **Hotspot mode** lets Ubuntu Desktop act as a temporary internet gateway.
- **Home, campus, and office networks behave differently**, so desktop connectivity choices should adapt to the environment.

Part VIII — Security, Privacy, and Reliability

Chapter 17 — Securing Ubuntu Desktop

Security on Ubuntu Desktop is not a single feature. It is a **layered practice**. A secure desktop depends on strong passwords, safe session habits, disk protection, a properly configured firewall, trusted software sources, sandboxed applications, cautious browsing, and a realistic backup plan. The Ubuntu Academy outline makes this chapter especially important by placing **desktop security** among the heaviest focus areas, including **strong passwords, automatic locking, full-disk LUKS encryption, UFW, AppArmor, safe repositories, Snap/Flatpak sandboxing, malware scanning, phishing awareness, and backup encryption**.

The attached references support this direction very well. **Ubuntu Linux Bible** explains password quality controls, UFW basics, AppArmor status and enforcement, encryption tools, and ClamAV. **Ubuntu: The Complete Guide** emphasizes why Linux is harder for malware to compromise than many consumer systems, highlights the importance of centralized repositories, warns against untrusted downloads and random terminal commands, and discusses encrypted systems, private home folders, and backup encryption. Together, they provide a strong foundation for a practical Ubuntu Desktop security chapter.

A useful illustration is to think of desktop security like protecting a house. A strong password is the front-door lock. Screen locking is closing the door when you step out. Full-disk encryption is the safe built into the floor. The firewall is the fence and gate. AppArmor is the rulebook that tells each guest which rooms they may enter. Safe software installation is choosing trustworthy builders and suppliers. Backups are the emergency plan if something still goes wrong.

Strong Passwords and Account Protection

The first line of defense on Ubuntu Desktop is still the user account. The Ubuntu Linux Bible explains that `passwd` checks password quality and that password rules can be enforced with **libpwquality** through `/etc/security/pwquality.conf`. It even gives a concrete example requiring long passwords with mixed character classes and reuse controls.

The same reference also notes that older Linux security policies often forced regular password changes, but that current best practice is more nuanced: frequent forced changes do not automatically improve security if users respond by choosing weak predictable patterns. That is a very useful lesson for desktop readers. A strong password policy should aim for **quality, uniqueness, and safe storage**, not just frequent change for its own sake.

Ubuntu: The Complete Guide adds a very practical desktop perspective: the biggest real-world threat for average users is often **online account compromise through**

weak or reused passwords. It strongly recommends password managers and even discusses two-factor authentication as a better modern habit than reusing human-memorable passwords across services.

A good desktop security habit is:

- use **long, unique passwords**
- do not reuse the same password across websites and local accounts
- use a **password manager**
- enable **two-factor authentication** where possible
- change passwords immediately if compromise is suspected

A simple illustration: a strong password is not just a lock. It is a lock that is different on every door.

CLI example: change the current user password

```
passwd
```

Example output:

```
Changing password for student.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

CLI example: inspect password-aging settings for a user

```
sudo chage -l student
```

Example output:

```
Last password change                                    : Mar 14, 2026
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
```

The Ubuntu Linux Bible explicitly uses `chage -l` to inspect and manage password aging.

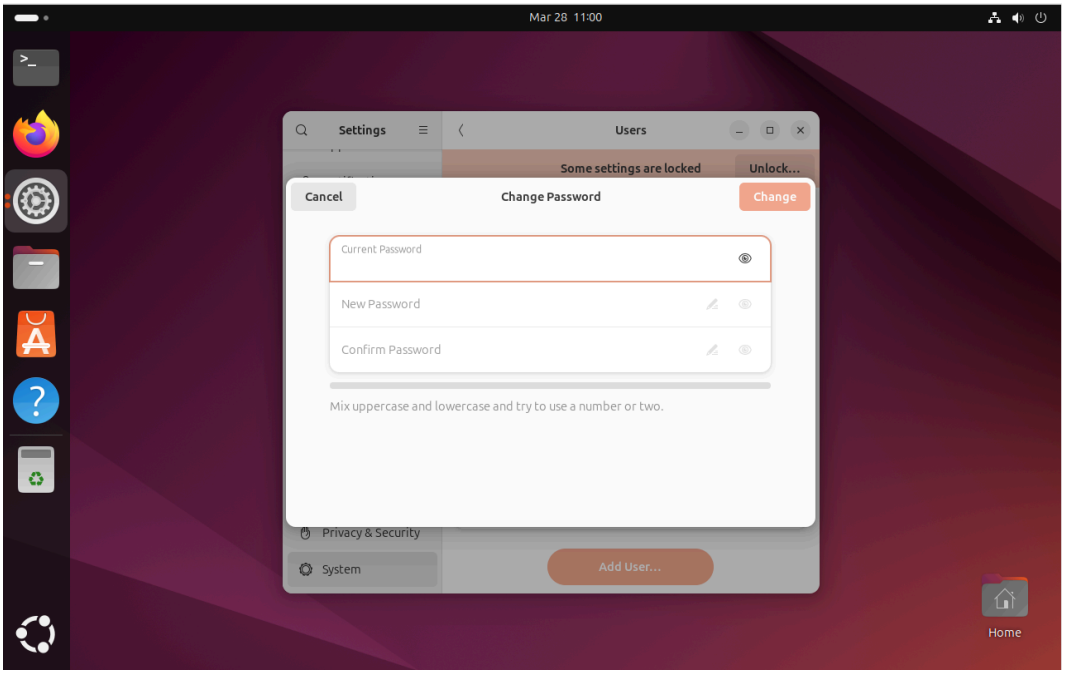


Figure. Settings > Users showing password change options for the current desktop user

Screen Locking and Session Security

A secure desktop is not only about passwords at login. It is also about **what happens after login**. The Ubuntu Academy outline explicitly includes **automatic locking** and broader session-security behavior.

On a desktop or laptop, one of the simplest and most effective habits is to **lock the screen whenever stepping away**. This matters in classrooms, offices, shared homes, labs, and public spaces. If the system is already unlocked, even the strongest password is no longer helping.

A good session-security mindset includes:

- lock the screen when leaving the desk
- do not leave sensitive files open on shared machines
- log out on public or shared systems
- avoid storing sensitive credentials in unsafe visible places
- prefer your own account over shared accounts

The broader Ubuntu references reinforce this by noting that user home folders are private by default in newer Ubuntu behavior and that Linux security benefits from standard users not having automatic power over the whole system.

A practical illustration is to think of session security like walking away from your office desk. The room may be in a secure building, but the papers on the table still matter.

CLI example: lock the current session

```
loginctl lock-session
```

Typical result:

```
(no output if successful)
```

CLI example: show the current desktop session type

```
echo $XDG_CURRENT_DESKTOP
echo $DESKTOP_SESSION
```

Example output:

```
ubuntu:GNOME
ubuntu
```

These commands are simple, but they help readers connect visible session behavior to the actual desktop session they are securing.

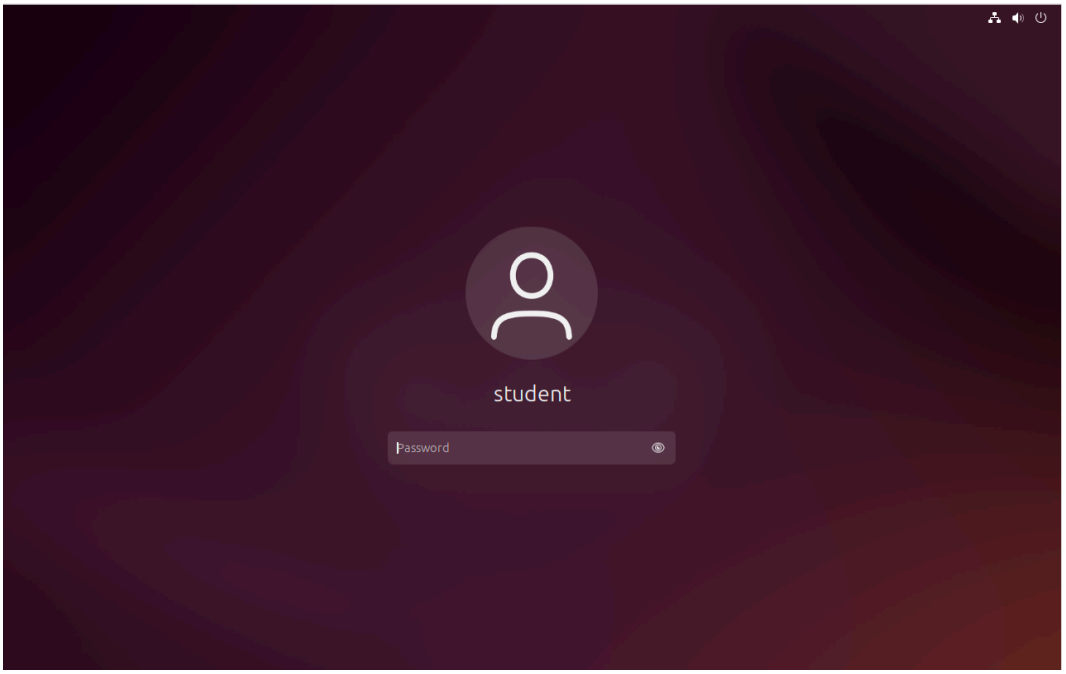


Figure. Lock screen after stepping away from an Ubuntu desktop session

Full-Disk Encryption Basics

Full-disk encryption is one of the strongest protections available for a desktop or laptop because it protects data **at rest**. The Ubuntu Academy outline explicitly names **full-disk LUKS encryption** as part of desktop system security.

The attached references support this clearly. The Ubuntu outline places encryption directly in the installation and security workflow, while *Ubuntu: The Complete Guide* notes that encrypted installations gained a useful recovery-key option and that encryption during installation is especially valuable if something goes wrong or the device is lost. It also emphasizes that private home folders and stronger storage protection meaningfully improve user privacy.

A beginner-friendly explanation is this: full-disk encryption means that if someone physically steals the machine, they cannot simply remove the drive, connect it elsewhere, and read your files as ordinary data. Without the unlock secret, the stored content remains protected.

This is especially important for:

- laptops
- portable desktops
- systems containing student records, research, or work documents
- travel devices
- machines stored in shared or semi-public environments

A good illustration is to think of encryption as turning the whole filing cabinet into a locked safe rather than only putting locks on individual folders.

CLI example: view block devices and encryption-related layout

```
lsblk -f
```

Example output:

```
NAME        FSTYPE      LABEL   UUID                   MOUNTPOINTS
nvme0n1
├─nvme0n1p1 vfat                1234-ABCD              /boot/efi
└─nvme0n1p2 crypto_LUKS         9f4a2d91-2e22-4b5f-a1b0
  └─cryptroot ext4              3b6c7a16-5cfd-4bdb-ae8f /
```

This kind of output helps learners visually understand the relationship between an encrypted container and the unlocked root filesystem.

CLI example: confirm the root device in use

```
mount | grep ' on / '
```

Example output:

```
/dev/mapper/cryptroot on / type ext4 (rw,relatime)
```

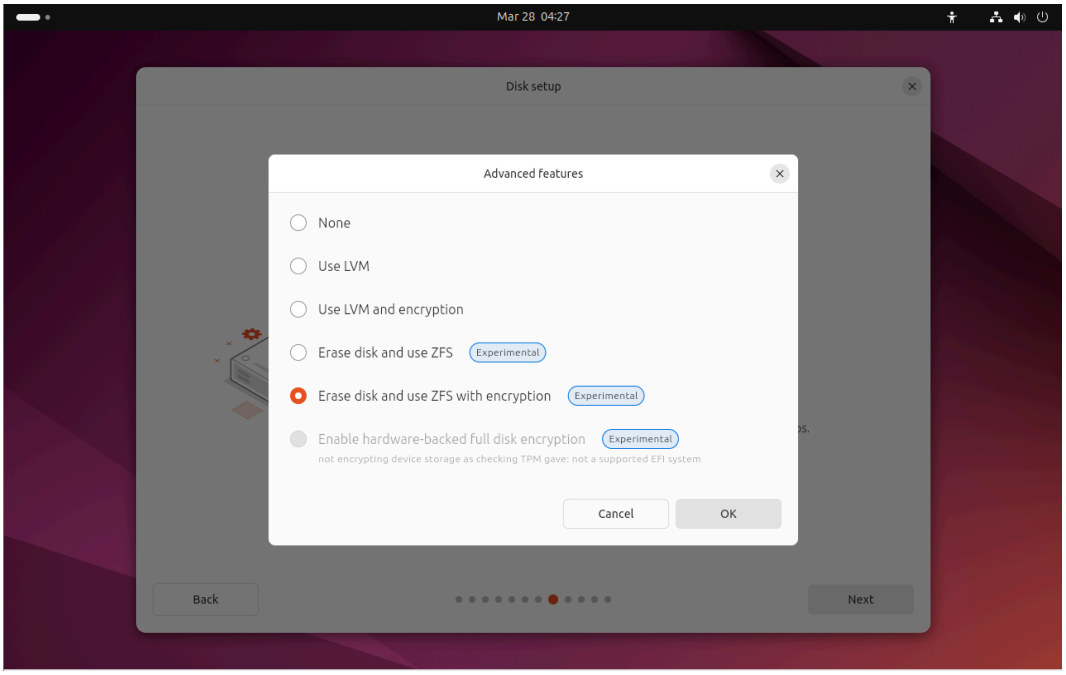


Figure. Ubuntu installer screen showing the option to enable disk encryption

Firewall with UFW

The Ubuntu Linux Bible provides strong, desktop-friendly guidance for **UFW**, Ubuntu's **Uncomplicated Firewall**. It explains that UFW is the usual Ubuntu front end for the Linux firewall stack and shows core actions such as checking status, allowing specific ports, limiting SSH access by subnet, enabling the firewall, and disabling it again if no longer needed. It also notes that Ubuntu can use the more graphical **gufw** front end for users who prefer a GUI.

This is highly relevant for Ubuntu Desktop because a personal workstation is still a **host** on a network. Even if a router provides some protection, a host-based firewall adds another security layer.

A practical desktop use of UFW may include:

- checking whether the firewall is active
- enabling it only after needed rules are understood
- allowing only specific services, such as SSH, VNC, or a temporary test service
- avoiding accidental overexposure of desktop-sharing or remote-access ports

A useful illustration is to think of UFW as the controlled entry gate to your desktop. It does not care what the building looks like; it cares who is allowed through the door.

CLI example: check firewall status

```
sudo ufw status
```

Example output:

```
Status: inactive
```

This exact pattern is shown in the Ubuntu Linux Bible.

CLI example: allow SSH only from the local subnet

```
sudo ufw allow from 192.168.1.0/24 to any port 22 proto tcp
```

Example output:

```
Rules updated
```

CLI example: enable the firewall

```
sudo ufw enable
```

Example output:

```
Command may disrupt existing ssh connections. Proceed
with operation (y|n)? y
Firewall is active and enabled on system startup
```

CLI example: check the active rules

```
sudo ufw status
```

Example output:

```
Status: active
To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       192.168.1.0/24
80                         ALLOW       Anywhere
80 (v6)                    ALLOW       Anywhere (v6)
```

This is closely aligned with the Bible's own example flow.

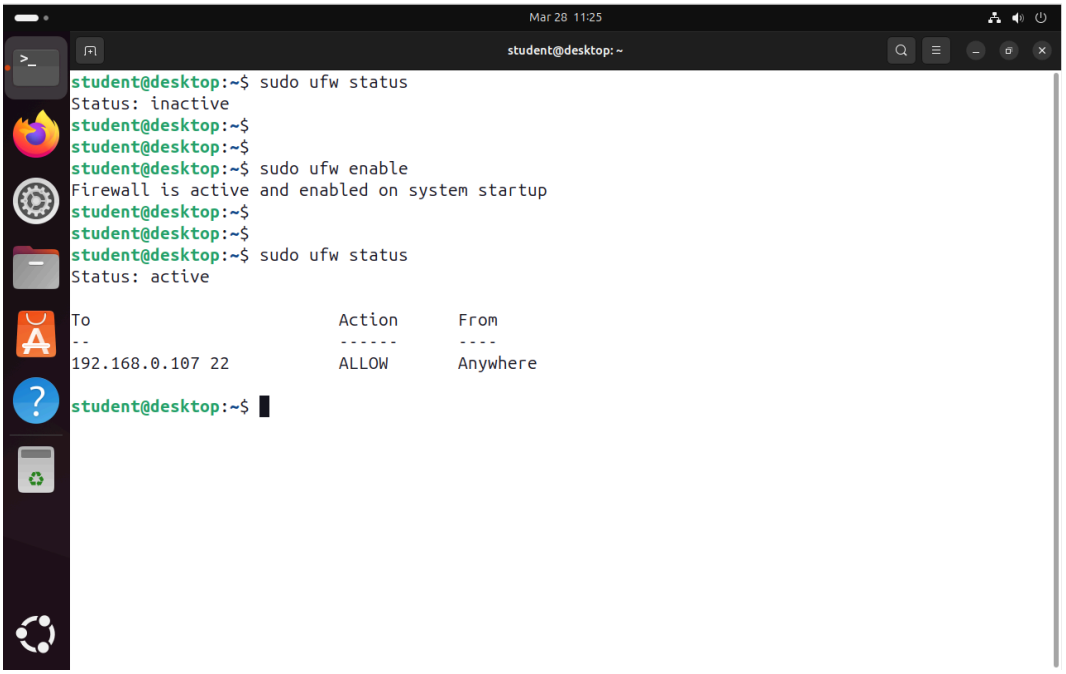


Figure. Terminal showing sudo ufw status before and after enabling the firewall

AppArmor Basics on Ubuntu

AppArmor is one of Ubuntu's important built-in security layers. The Ubuntu Academy outline explicitly lists **AppArmor basics** under desktop security.

The Ubuntu Linux Bible explains AppArmor in practical terms by showing how profiles can be loaded, placed in **enforce** mode, and checked with `apparmor_status`. Although the example in the book uses Apache, the important idea for Ubuntu Desktop readers is much broader: AppArmor applies **policy-based restrictions** to programs so that even if an application misbehaves, it may be limited in what it can access or change.

A beginner-friendly explanation is to think of AppArmor as giving each application a map with approved rooms. The application may move around, but only within the boundaries allowed by its profile.

This matters because desktop applications increasingly handle:

- downloaded files
- browser content
- untrusted media
- network connections
- local secrets or cloud credentials

A practical security lesson is that Linux security is not only about whether software runs. It is also about what that software is *allowed* to do.

CLI example: check AppArmor status

```
sudo apparmor_status
```

Example output:

```
141 profiles are loaded.
46 profiles are in enforce mode.
0 profiles are in complain mode.
```

The Bible snippet shows this style of output directly.

CLI example: inspect AppArmor package presence

```
dpkg -l | grep apparmor
```

Example output:

```
ii  apparmor           4.0.1-0ubuntu1   amd64
user-space parser utility for AppArmor
ii  apparmor-utils     4.0.1-0ubuntu1   amd64
utilities for AppArmor profiles
```

CLI example: view active AppArmor-related lines

```
sudo apparmor_status | head
```

Example output:

```
141 profiles are loaded.
```

```
46 profiles are in enforce mode.
   /usr/bin/man
   /usr/lib/snapd/snap-confine
```

This also connects naturally with Ubuntu's Snap security model, because snap-confine appears as part of the enforced AppArmor ecosystem in the reference material.

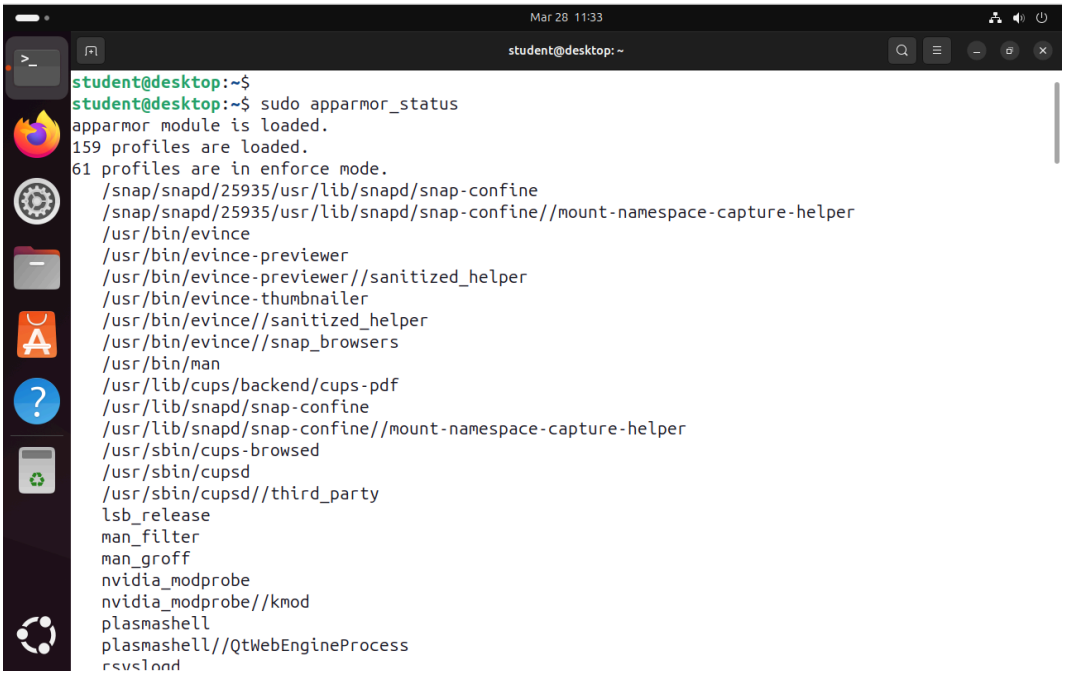


Figure. Terminal showing apparmor_status output on Ubuntu

Trusted Repositories and Safe Software Installation

One of Ubuntu's biggest security strengths is that software is usually installed from **centralized, managed repositories**. *Ubuntu: The Complete Guide* makes this point very clearly: many Windows infections begin with software downloaded from untrustworthy websites, whereas Ubuntu and most Linux distributions rely primarily on centralized software repositories.

The Ubuntu Linux Bible adds the essential caution: if you add PPAs or third-party repositories, you must **trust them**, because they are not curated and scanned in the same way as the main repositories. It also points out that if repository keys cannot be confirmed, then the guarantee of package integrity is weakened and the software should not be trusted.

This is one of the most important real-world security lessons in Ubuntu Desktop administration:

- prefer official repositories first
- treat PPAs and third-party repositories with caution
- remove broken or outdated repository entries
- do not install random `.deb` packages just because a web page says so
- think about software trust before thinking about software convenience

A good illustration is to compare official repositories to a university library and unknown PPAs to books handed to you in the parking lot. Some may be fine, but the trust level is not the same.

CLI example: refresh repository metadata

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
```

CLI example: add a PPA carefully

```
sudo apt-add-repository ppa:example/project
```

Example output:

```
Press [ENTER] to continue or Ctrl-c to cancel adding it.
```

CLI example: remove an untrusted or outdated source file

```
ls /etc/apt/sources.list.d/
sudo rm /etc/apt/sources.list.d/example-project-noble.list
sudo apt update
```

This closely follows the cleanup pattern described in the Ubuntu Linux Bible.

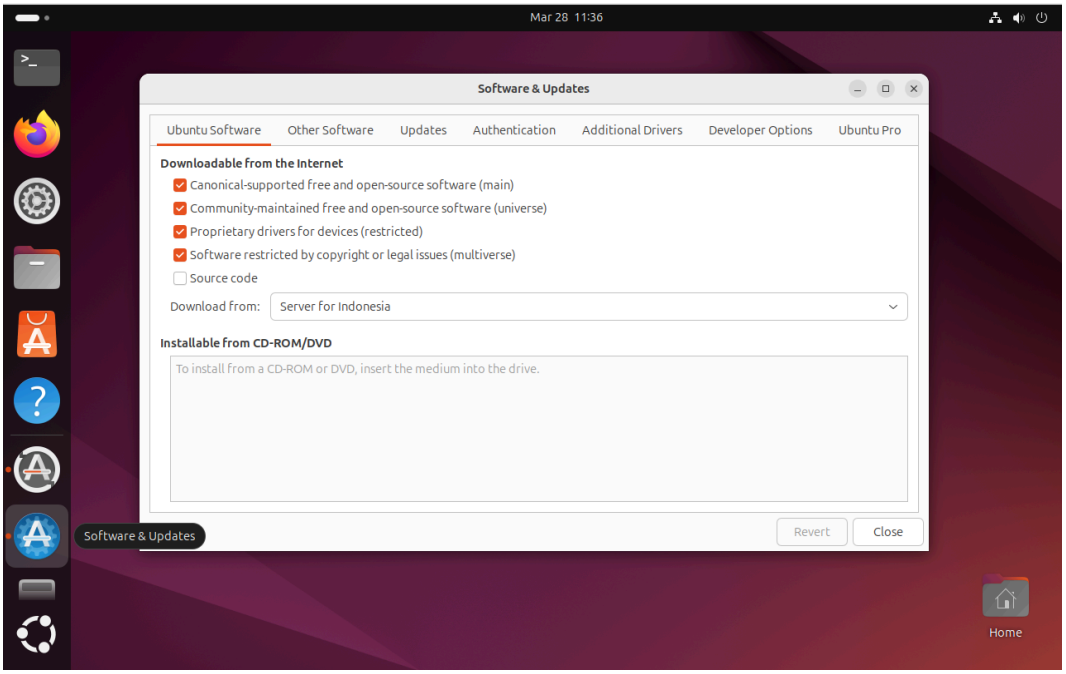


Figure. Software & Updates > Other Software tab showing repository entries

Snap/Flatpak Sandboxing Benefits

The Ubuntu Academy outline explicitly includes **browser and app sandboxing** and highlights the benefits of **Snap** and **Flatpak**.

The attached references support this in two complementary ways. **Mastering Ubuntu** explains that Snap packages are self-contained and regularly updated with security and bug fixes, though they may use more space and start more slowly than standard `deb` packages. The Ubuntu Linux Bible shows AppArmor profiles related to `snap-confine`, which underlines that the Ubuntu desktop security model increasingly includes containment and policy-based restriction around packaged applications.

A practical security explanation is:

- **traditional packages** integrate deeply with the system
- **sandboxed packages** try to reduce unrestricted access to the host
- **self-contained packaging** can reduce dependency confusion and simplify updates
- the security benefit is not absolute, but sandboxing is a useful extra barrier

A good illustration is to think of Snap and Flatpak apps as working inside managed rooms with controlled doors rather than having free access to the whole building.

CLI example: list installed snaps

```
snap list
```

Example output:

```
Name     Version   Rev   Tracking       Publisher
Notes
core22   20250214  1880  latest/stable  canonical✓
base
snapd    2.67      23545 latest/stable  canonical✓
snapd
firefox  136.0     6014  latest/stable  mozilla✓    -
```

CLI example: install a Snap application

```
sudo snap install foliate
```

Example output:

```
foliate 3.2.1 from community installed
```

This aligns with the Snap workflow discussed in Mastering Ubuntu.

CLI example: verify Flatpak support

```
flatpak remotes
```

Example output:

```
flathub    system
```

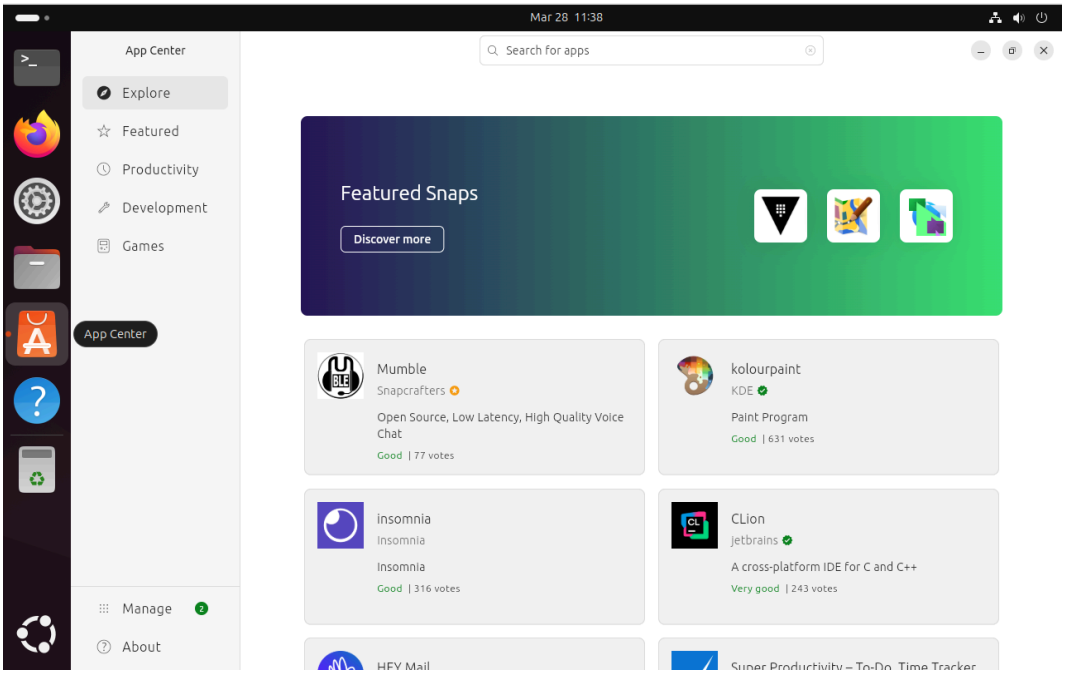


Figure. App Center showing an application source identified as Snap

Browser Security, Downloads, and Phishing Awareness

The desktop browser is one of the highest-risk entry points for ordinary users, not because Ubuntu is weak, but because the browser is where users meet the open internet. *Ubuntu: The Complete Guide* is especially strong here. It says that Linux users are still vulnerable to **phishing** and other forms of online social engineering and explicitly advises users to:

- keep the system up to date
- not download software from untrusted sources
- not run terminal commands from random sources
- stay cautious online
- use an up-to-date browser

This is exactly the right desktop-security mindset. Most average users are more likely to be harmed by **deception** than by a classic self-spreading Linux virus.

A practical browser-security checklist includes:

- verify websites before entering credentials
- avoid suspicious extensions
- prefer HTTPS and known domains
- do not download random install scripts
- do not paste unexplained commands into Terminal
- keep the browser and system updated

A useful illustration is to think of phishing as someone wearing a fake uniform at the gate. The threat is not always technical sophistication. Sometimes it is simply tricking the user into opening the wrong door.

CLI example: verify that the browser package is up to date through APT metadata

```
apt list --upgradable | grep firefox
```

Example output:

```
firefox/noble-updates 1:1snap1-0ubuntu2 amd64
[upgradable from: 1:1snap1-0ubuntu1]
```

CLI example: inspect saved passwords and keys tool availability

```
which seahorse
```

Example output:

```
/usr/bin/seahorse
```

The Ubuntu Linux Bible explicitly discusses Passwords and Keys and launching it with seahorse to manage saved passwords, certificates, GPG keys, and SSH keys from the GNOME desktop.

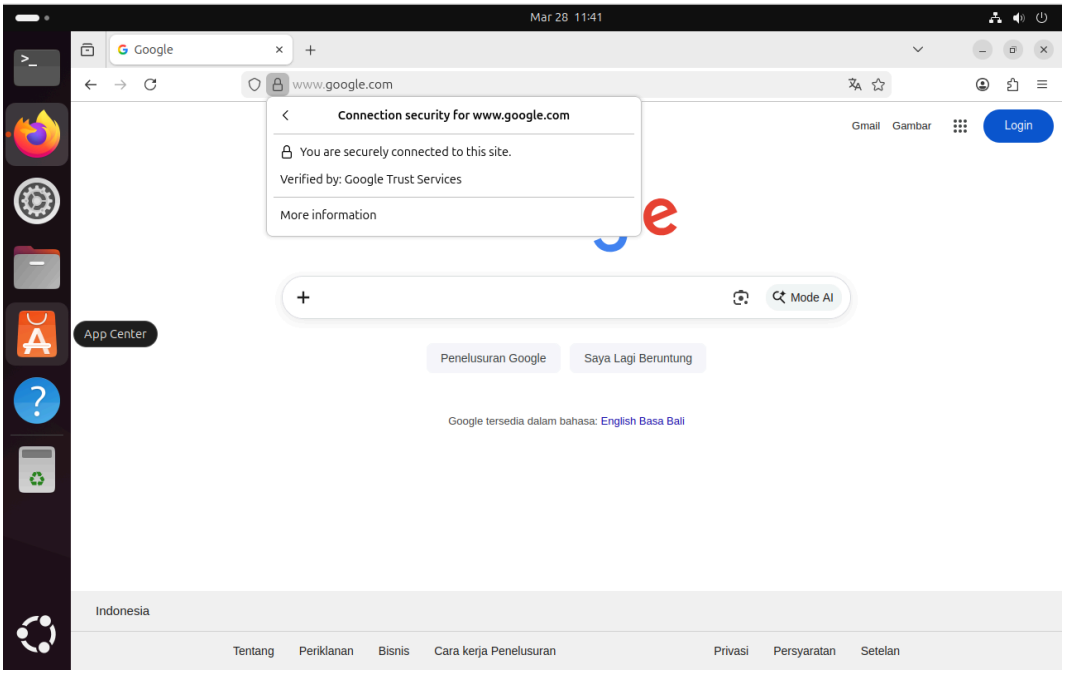


Figure. Browser showing a secure HTTPS site and privacy/security indicators.

Intro to Malware Scanning and Safe Habits

The attached references take a balanced view of Linux malware. *Ubuntu: The Complete Guide* explains that Linux desktops are generally much less attractive and less vulnerable to mass malware outbreaks than Windows consumer systems, partly because of permission design, repositories, and update behavior. But it also clearly says that Linux is **not magically immune** and that there are still valid reasons to use scanning tools, especially when interacting with Windows systems, shared drives, or business requirements.

The Ubuntu Linux Bible supports this realism. It states that it is a misconception that Linux has no viruses, explains why ClamAV is a good open-source antivirus choice, and also introduces rootkit detection tools such as `chkrootkit`.

A good desktop-security lesson is:

- Ubuntu is **safer by design**, but not invincible
- malware scanning can still be useful
- the biggest daily risks are often **unsafe downloads, weak passwords, and phishing**
- antivirus is a tool, not a replacement for judgment

CLI example: install ClamAV

```
sudo apt install clamav clamav-freshclam
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  clamav clamav-freshclam
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
```

The Ubuntu Linux Bible gives this exact package pair.

CLI example: update or refresh signatures

```
sudo freshclam
```

Typical result:

```
ClamAV update process started
daily database available for download
```

CLI example: scan the home directory

```
clamscan -r /home/student
```

Example output:

```
----------- SCAN SUMMARY -----------
Known viruses: 8734212
Engine version: 1.3.1
```

```
Scanned directories: 218
Scanned files: 1432
Infected files: 0
```

The Complete Guide also recommends `clamscan -r /home` for recursive scanning and notes that `clamtk` can provide a GUI front end.

CLI example: install a rootkit checker

```
sudo apt install chkrootkit
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
```

The following NEW packages will be installed:

```
  chkrootkit
```

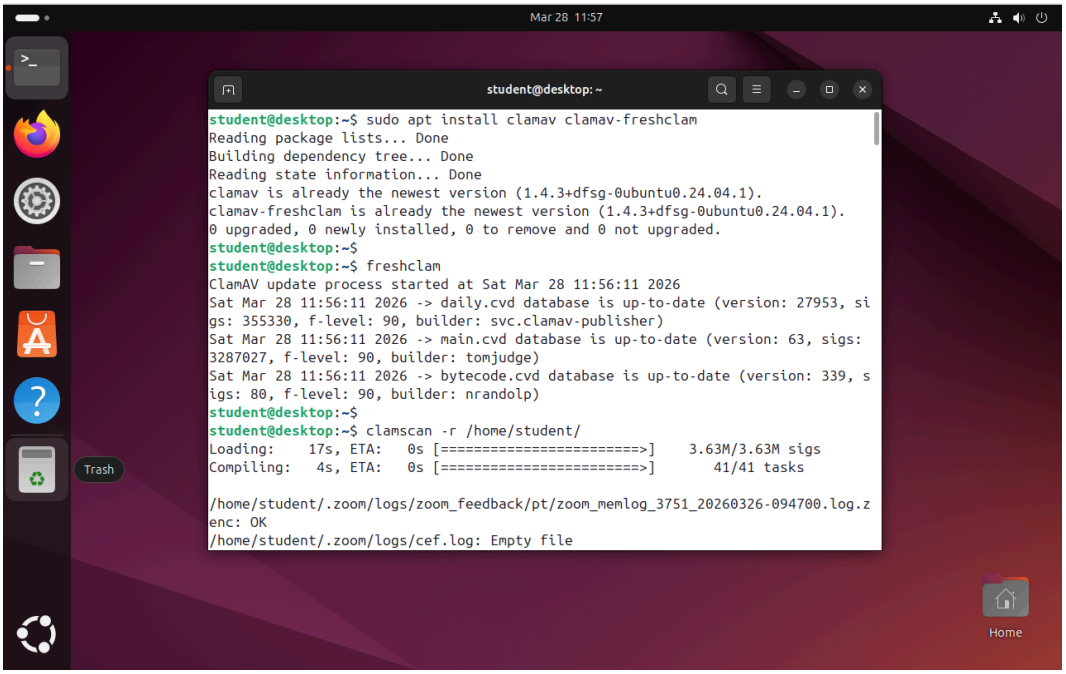


Figure. Terminal showing ClamAV installation and a scan summary

Backup Encryption and Recovery Planning

Security is incomplete without recovery. The Ubuntu Academy outline explicitly includes **backup encryption and recovery planning**, which is exactly right: a secure desktop must survive mistakes, theft, corruption, and device failure—not just resist attack.

Ubuntu: The Complete Guide provides especially useful support here. It explains that **Déjà Dup** backups are incremental and that online backups, such as FTP or cloud-synced storage, should be **encrypted with a password**, especially if the data is sensitive. It also notes that Ubuntu's backup tools can remember credentials through the password manager.

The Ubuntu Linux Bible broadens the picture by listing tools such as `duplicity` for encrypted backups and `seahorse` for password/key management on the GNOME desktop.

A practical backup-security mindset includes:

- back up important files regularly
- encrypt backups, especially if stored online or on portable media
- test that restore actually works
- store recovery information safely
- keep one backup separate from the main machine

A useful illustration is to think of backup encryption like storing a spare house key in a locked emergency box instead of taping it to the front door.

CLI example: install Déjà Dup if needed

```
sudo apt install deja-dup
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
```

The following NEW packages will be installed:

```
  deja-dup
```

CLI example: install duplicity for encrypted backup workflows

```
sudo apt install duplicity
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
```

The following NEW packages will be installed:

```
  duplicity
```

The Ubuntu Linux Bible explicitly describes `duplicity` as a tool that encrypts backups.

CLI example: launch the GNOME password/key manager

```
seahorse &
```

Example output:

```
[1] 7420
```

This supports the desktop-side management of saved secrets related to encryption and secure services.

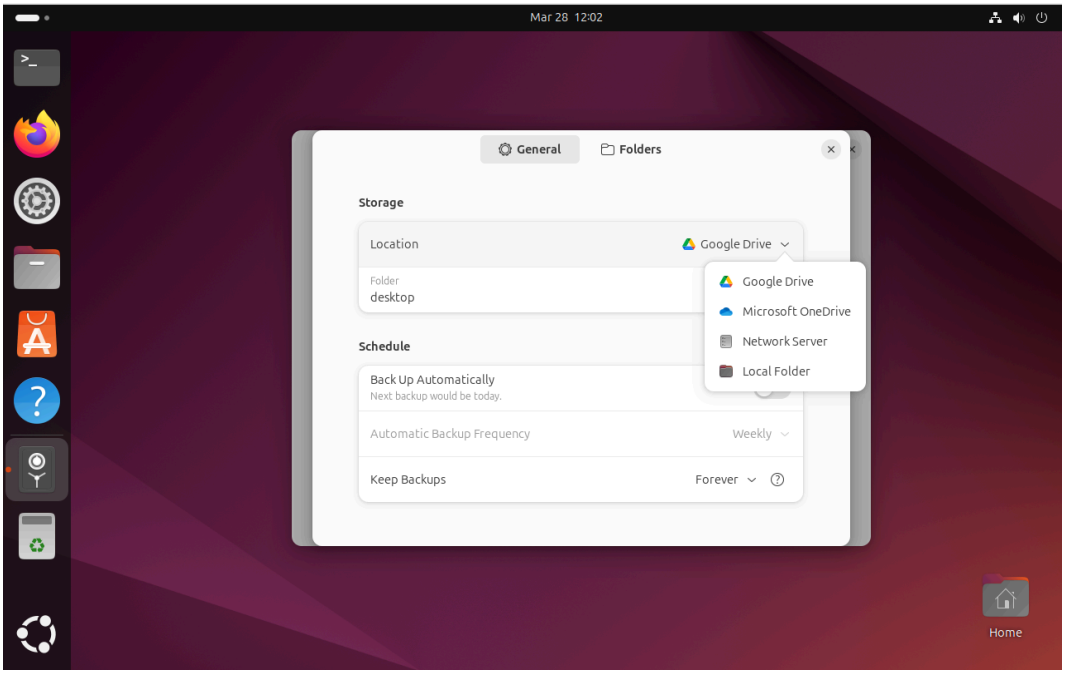


Figure. Déjà Dup backup window showing encrypted backup configuration

Guided Desktop Security Lab

To make this chapter practical, readers can complete a short hands-on security exercise:

- change the account password with `passwd`
- inspect password-aging information with `chage -l`
- lock the session and sign back in
- check UFW status, add one safe rule, and enable the firewall
- run `apparmor_status`
- inspect repository entries and identify whether any unnecessary third-party source is present
- install and run a small Snap or Flatpak app and note its sandboxed packaging model
- install ClamAV and scan the home directory
- install Déjà Dup or duplicity and prepare an encrypted backup plan
- open Seahorse and review saved keys/passwords

This kind of lab turns security from theory into repeatable habit.

Chapter Reflection

Securing Ubuntu Desktop is best understood as a layered system of habits and controls. Strong passwords and careful account protection reduce the chance of unauthorized access. Screen locking and good session hygiene protect active desktops. Full-disk encryption protects stored data if the device is lost or stolen. UFW adds a host-based firewall layer, while AppArmor helps confine what applications are allowed to do. Official repositories and cautious software installation habits reduce supply-chain risk, and sandboxed packaging through Snap or Flatpak adds another useful boundary. Browser security, phishing awareness, and careful download behavior matter because many real attacks target users rather than kernels. Malware scanning with tools such as ClamAV can still be useful, especially in mixed environments. Finally, encrypted backups and realistic recovery planning ensure that security includes not only prevention, but also survival and recovery.

Key Takeaways

- **Ubuntu Desktop security is layered, not dependent on one feature alone.**
- **Strong, unique passwords and password managers** are more effective than weak recycled credentials.
- **Full-disk encryption** protects data at rest, especially on laptops and portable systems.
- **UFW** is Ubuntu's practical firewall front end and is suitable for desktop host protection.
- **AppArmor** adds application confinement and is an important Ubuntu security layer.
- **Official repositories are safer than random downloads or untrusted PPAs.**
- **Snap/Flatpak sandboxing** provides useful isolation benefits for desktop apps.
- **Phishing and unsafe commands/downloads** are often more realistic threats than classic Linux viruses.
- **ClamAV and encrypted backups** are useful tools when part of a broader safe-desktop practice.

Chapter 18 — Updates, Maintenance, and Backup

A healthy Ubuntu Desktop is not a machine that is installed once and then forgotten. It is a system that is **updated carefully, maintained regularly, cleaned when necessary, and backed up before problems become disasters**. In the Ubuntu Academy outline, this chapter is directly tied to **automatic updates, manual update workflows, unattended-upgrades, cache cleanup, backup and restore basics with Déjà Dup, and snapshot concepts with Timeshift**.

The attached references support this chapter very well from different angles. *Ubuntu: The Complete Guide* explains that Ubuntu's desktop update model is more considerate than forced consumer-style updates and notes that **urgent fixes can be applied in the background** while general package updates remain under the user's control. It also gives practical guidance for **Déjà Dup backups**, including incremental backups, restore previews, and encryption for online backups. *Ubuntu Linux Bible* emphasizes the importance of keeping system software current, recommends **unattended-upgrades** for efficient patching, and reinforces the core APT workflow for updates.

A useful illustration is to think of Ubuntu Desktop as a car. **Updates** are routine servicing, **maintenance** is keeping the vehicle clean and efficient, **snapshots** are like taking a full state photo before major changes, and **backups** are the emergency spare parts and document copies you keep in case the vehicle is stolen, damaged, or misconfigured.

Manual and Automatic Updates

Ubuntu Desktop supports both **manual** and **automatic** update workflows. This balance is one of its strengths. *Ubuntu: The Complete Guide* explicitly says that Ubuntu uses an **unattended upgrade service** to apply urgent fixes in the background, while general package updates are presented to the user and can be applied at a convenient time from the Software application or from the command line. That is an important desktop lesson: Ubuntu updates can be both **responsible** and **user-friendly**.

The Ubuntu Academy outline reinforces this by explicitly naming **manual update workflows: GUI Software Updater vs command-line** and **automatic updates: unattended-upgrades configuration**.

A practical distinction is:

- **manual updates** are ideal when the user wants to review changes first
- **automatic updates** are ideal for urgent security fixes and lower-friction maintenance
- together they form a reasonable desktop patching strategy rather than an all-or-nothing choice

A useful analogy is this: manual updates are like checking the maintenance schedule yourself, while automatic updates are like having the building manager quietly replace a broken safety light before anyone gets hurt.

CLI example: refresh package metadata

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
```

The Ubuntu Linux Bible explicitly describes apt update as polling remote repositories for recent changes and updating the local package index.

CLI example: apply package upgrades

```
sudo apt upgrade -y
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Calculating upgrade... Done
The following packages will be upgraded:
  firefox linux-firmware
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Mastering Ubuntu also explains that sudo apt upgrade installs upgrades for packages already present on the system after package lists have been refreshed.

CLI example: list upgradeable packages

```
apt list --upgradable
```

Example output:

```
Listing... Done
firefox/noble-updates 1:1snap1-0ubuntu2 amd64
[upgradable from: 1:1snap1-0ubuntu1]
```

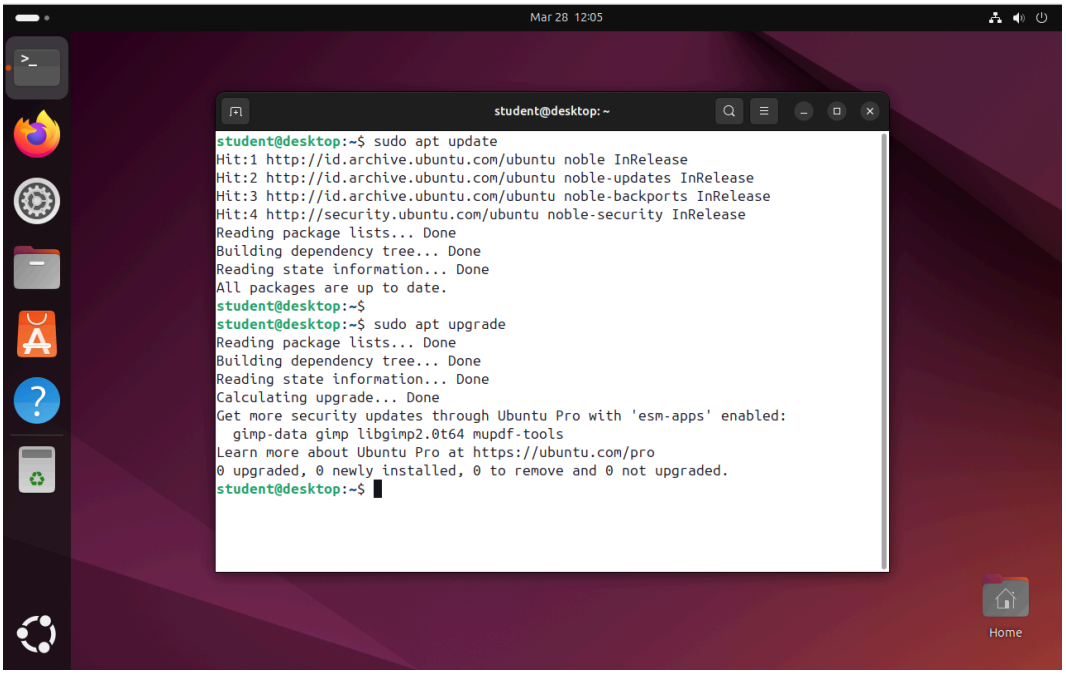


Figure. Terminal running sudo apt update and sudo apt upgrade

Security Updates and Patch Strategy

Not every update has the same urgency. A good desktop administrator develops a **patch strategy**, not just a habit of clicking “Update” blindly. The Ubuntu Linux Bible is direct on this point: when security holes are discovered and patched, the updated packages need to be installed, and the safest, most efficient answer is to apply APT updates promptly. It specifically recommends automating updates with a tool like **unattended-upgrades**.

Ubuntu: The Complete Guide complements this by describing Ubuntu’s approach as more measured than disruptive, with urgent fixes in the background and broader package changes offered at a convenient time.

A practical patching strategy for Ubuntu Desktop usually looks like this:

- install **security updates quickly**
- apply general package updates regularly, but at a sensible time
- do major release upgrades more carefully than ordinary package updates
- back up or snapshot important systems before bigger changes
- read warnings if updates affect graphics, kernels, or boot components

A good illustration is to think of patching like building maintenance. If there is a broken lock on the front door, fix it immediately. If the paint on the hallway walls is being refreshed, schedule that at a convenient time.

CLI example: inspect the current Ubuntu release-upgrade prompt policy

```
cat /etc/update-manager/release-upgrades
```

Example output:

```
Prompt=lts
```

Ubuntu: The Complete Guide specifically mentions editing this file to switch between LTS-only and broader release-notification behavior.

CLI example: upgrade to a newer release when appropriate

```
sudo do-release-upgrade
```

Typical result:

```
Checking for a new Ubuntu release
Please install all available updates for your release
before upgrading.
```

This is a good reminder that release upgrades are a separate class of maintenance task and should be approached more carefully than ordinary package updates.

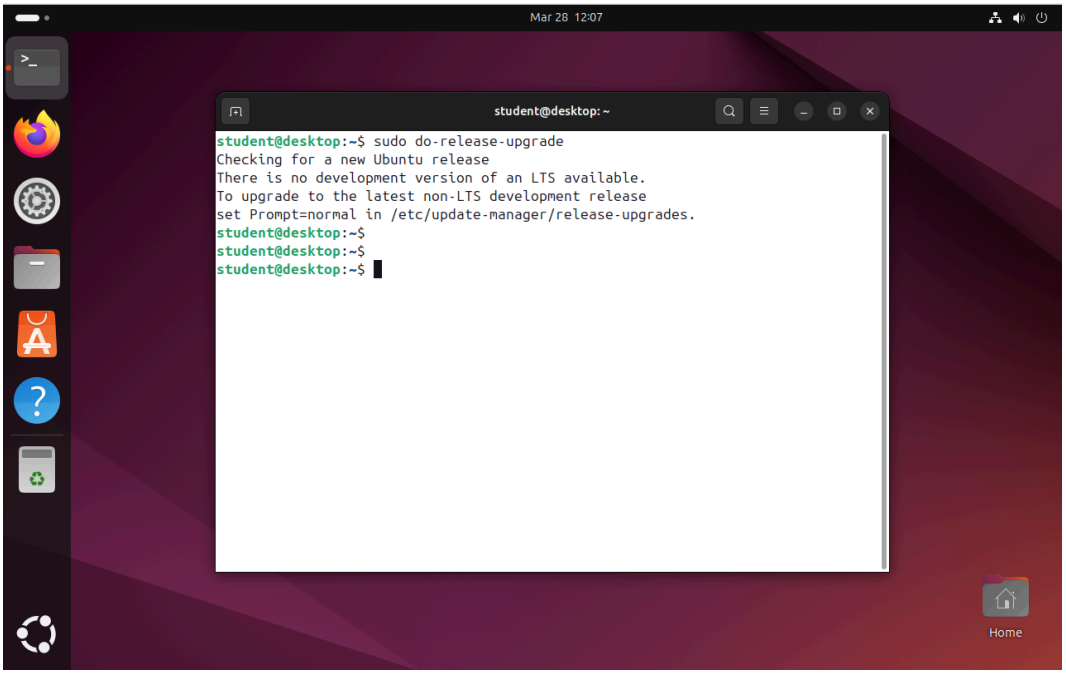


Figure. Terminal output from do-release-upgrade

unattended-upgrades Basics

The Ubuntu Academy outline explicitly calls out **unattended-upgrades configuration**, and both attached references support its role. *Ubuntu: The Complete Guide* says Ubuntu has an unattended upgrade service for urgent fixes in the background, while *Ubuntu Linux Bible* explicitly recommends installing and using `unattended-upgrades` as the efficient way to keep systems patched.

This is especially useful on desktops that should stay safe with minimal friction, such as:

- student laptops
- office workstations
- shared desktops
- family machines
- support-sensitive systems that should receive security fixes promptly

A beginner-friendly explanation is simple: **unattended-upgrades is like a quiet night-shift maintenance worker who applies urgent fixes without waiting for the user to remember every single one**.

CLI example: install unattended-upgrades

```
sudo apt install unattended-upgrades
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  unattended-upgrades
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

This matches the Ubuntu Linux Bible recommendation directly.

CLI example: check whether the service is present

```
systemctl status unattended-upgrades
```

Example output:

```
● unattended-upgrades.service - Unattended Upgrades
Shutdown
     Loaded: loaded
(/lib/systemd/system/unattended-upgrades.service;
enabled)
     Active: inactive (dead)
```

CLI example: review configuration file presence

```
ls /etc/apt/apt.conf.d/ | grep unattended
```

Example output:

```
20auto-upgrades
50unattended-upgrades
```

These files are central to how unattended upgrades behave on Ubuntu systems.

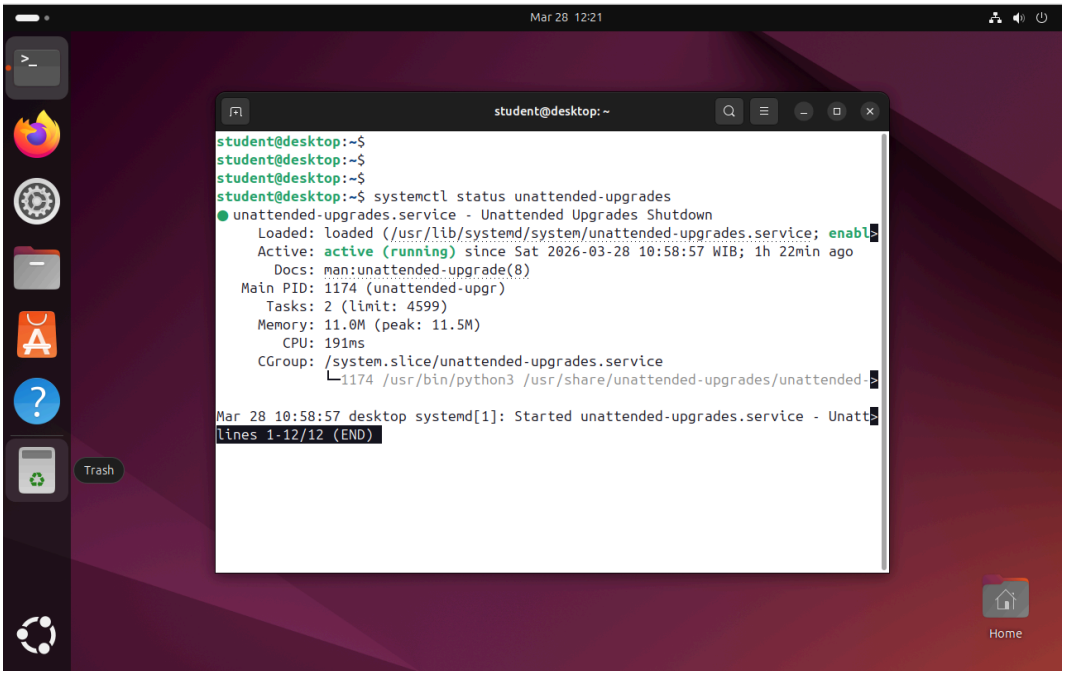


Figure. Terminal output show status unattended upgrades.

Routine System Maintenance

Updates are only part of maintenance. A healthy Ubuntu Desktop also needs simple, repeatable upkeep. The Ubuntu Academy outline treats this chapter as broader than patching alone, and the attached references support that wider view by emphasizing that Ubuntu desktops should be kept current, well-organized, and efficient over time.

A practical routine-maintenance checklist includes:

- checking for updates
- removing packages no longer needed
- making sure enough disk space remains available
- reviewing startup items if the system feels slow
- keeping browser and user-facing applications current
- confirming that backups still run and can still be restored

A useful illustration is to think of routine maintenance like caring for a study desk. If old papers, unused tools, and stale notes accumulate forever, even simple work becomes frustrating.

CLI example: check free disk space

```
df -h
```

Example output:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p2  236G   34G  191G  15% /
```

CLI example: inspect system uptime

```
uptime
```

Example output:

```
10:31:22 up 3:04,  2 users,  load average: 0.34, 0.41, 0.47
```

CLI example: list installed packages count roughly

```
dpkg -l | wc -l
```

Example output:

```
2143
```

This is not a direct “health” command, but it helps illustrate that desktop systems gradually accumulate software and complexity over time.

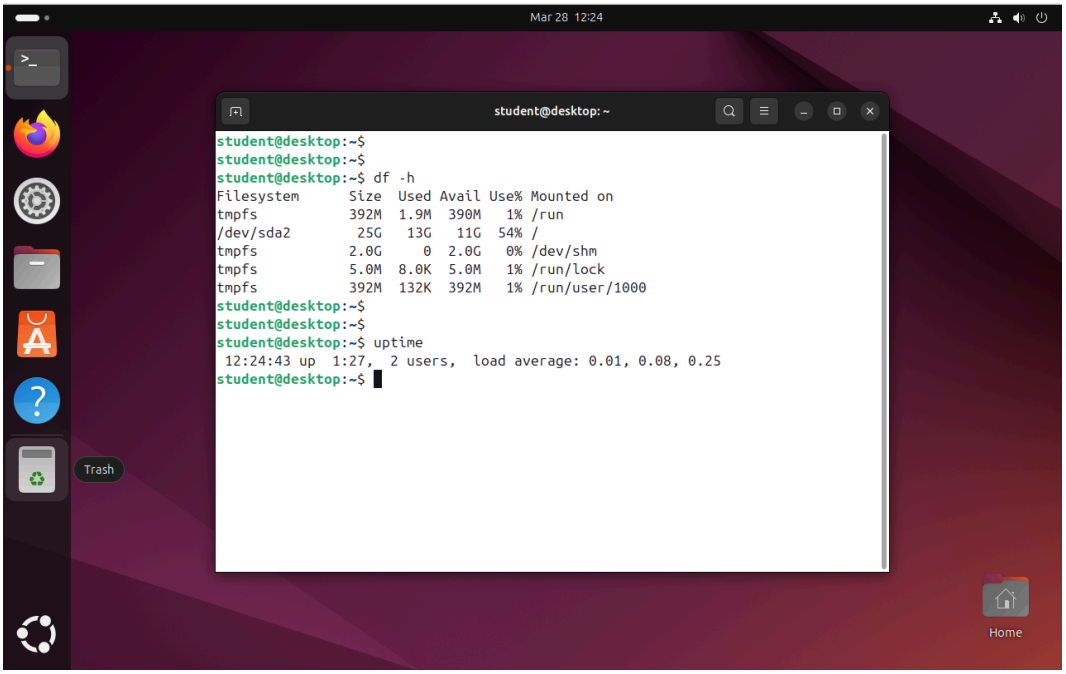


Figure. Terminal show Disk Usage and Uptime.

Cleaning Cache and Reducing Bloat

A desktop that is updated often and used daily will gradually collect **package cache files, old dependencies, and unused software**. The Ubuntu Academy outline explicitly mentions **performance tuning** and **clearing cache**, which makes cleanup part of responsible maintenance, not just a cosmetic extra.

The attached references do not present a large dedicated cleanup section, but they strongly support APT-based package management and update behavior, which naturally includes cleaning stale packages and reducing unnecessary package residue. The outline itself also directly points toward **backup and restore basics** in the same maintenance chapter, which reinforces the idea that cleanup should be done carefully, not recklessly.

A practical cleanup strategy includes:

- remove packages that are no longer needed
- clear APT cache when it grows unnecessarily
- avoid installing large duplicate app stacks without purpose
- archive old project files rather than leaving everything in active folders
- review Snap and Flatpak installations occasionally

A useful analogy is to think of cache cleanup like emptying storage boxes in a workshop. Not everything should be thrown away, but not everything should be kept forever either.

CLI example: remove automatically installed unused packages

```
sudo apt autoremove
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED:
  libexample1 old-runtime-helper
After this operation, 148 MB disk space will be freed.
```

CLI example: clean APT package cache

```
sudo apt clean
```

Typical result:

```
(no output if successful)
```

CLI example: inspect APT cache size before or after cleanup

```
sudo du -sh /var/cache/apt/archives
```

Example output:

```
24M   /var/cache/apt/archives
```

These are simple but effective maintenance habits for keeping a desktop lean without turning cleanup into an obsession.

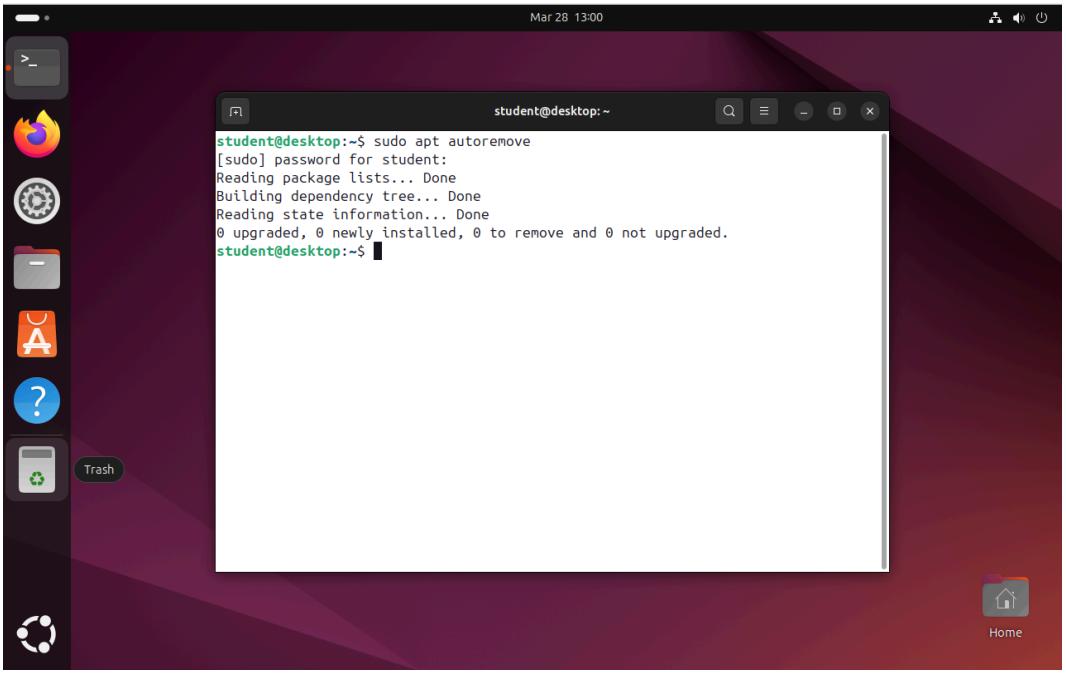


Figure. Terminal showing apt autoremove freeing disk space

Backup with Déjà Dup

Ubuntu: The Complete Guide gives especially strong support for **Déjà Dup**. It explains that backups made through Déjà Dup are **incremental**, meaning only files changed since the last backup are copied, which saves time and space. It also describes how the restore workflow presents a summary so users can confirm that they are restoring the correct backup, and it discusses both local and online backup targets.

This is exactly the kind of desktop backup tool that fits Ubuntu Academy's practical focus. Déjà Dup is valuable because it makes backup feel approachable rather than intimidating.

A strong beginner backup mindset is:

- back up personal files regularly
- keep at least one backup outside the main computer
- prefer automation over relying on memory alone
- encrypt sensitive backups
- occasionally test restore, not just backup creation

A good illustration is to think of Déjà Dup like a quiet librarian making copies of important notebooks every evening, but only copying the pages that changed.

CLI example: install Déjà Dup

```
sudo apt install deja-dup
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
```

The following NEW packages will be installed:

```
  deja-dup
```

CLI example: launch Déjà Dup

```
deja-dup &
```

Example output:

```
[1] 7562
```

CLI example: prepare a simple backup target folder

```
mkdir -p ~/Backups/DesktopBackup
ls ~/Backups
```

Example output:

```
DesktopBackup
```

This is a useful teaching step even when the actual backup workflow is driven mostly through the GUI.

\

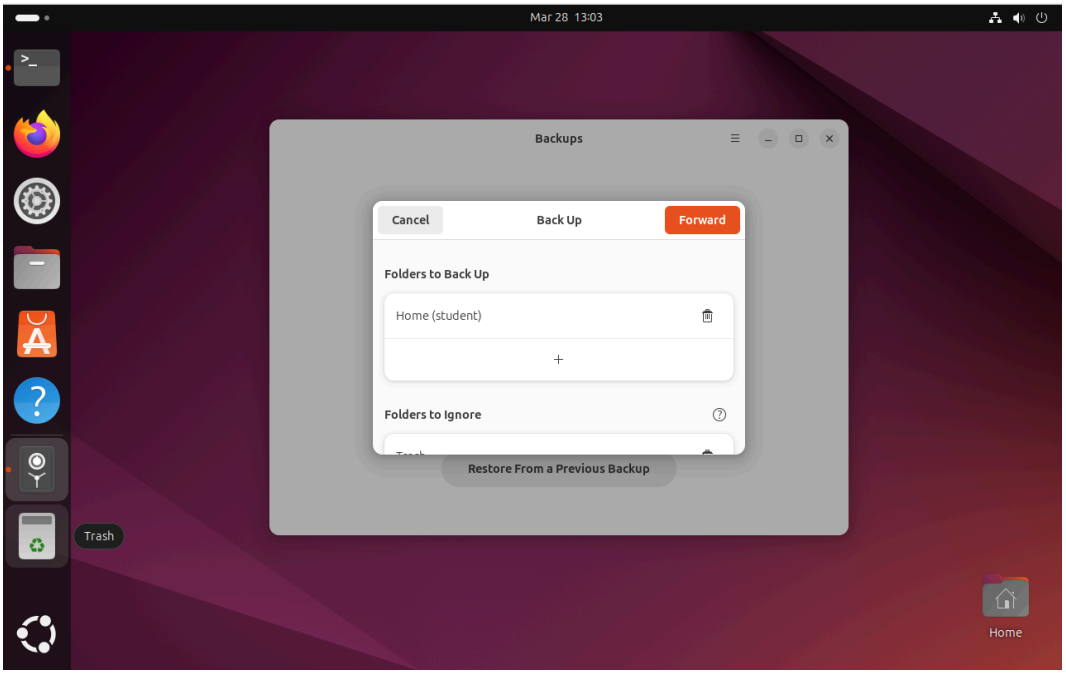


Figure.Déjà Dup main window showing folders to save and storage location

Snapshot Concepts with Timeshift

The Ubuntu Academy outline explicitly includes **Timeshift intro** under backup and restore basics. That is important because readers should understand that **file backups** and **system snapshots** are related, but not identical.

The attached reference set is much stronger on Déjà Dup than on Timeshift specifically, so the safest grounded treatment here is conceptual: Timeshift-style snapshots are useful for preserving a **system state** before risky changes such as:

- major updates
- driver changes
- desktop-environment experiments
- configuration surgery
- recovery-oriented maintenance

A practical explanation is:

- **file backup** protects documents, media, and personal data
- **snapshot** protects the state of the operating system and configuration at a point in time
- together they form a stronger recovery strategy than either one alone

A good illustration is to think of Déjà Dup as copying your books and papers, while a snapshot is photographing the whole room arrangement before you start moving furniture around.

CLI example: install Timeshift

```
sudo apt install timeshift
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
```

The following NEW packages will be installed:

```
  timeshift
```

CLI example: launch Timeshift

```
timeshift-launcher &
```

Example output:

```
[1] 7635
```

Because the attached books do not provide detailed Timeshift procedural content in the retrieved excerpts, the most responsible treatment is to introduce it as a snapshot concept and practical companion to file backup rather than overclaiming exact details.

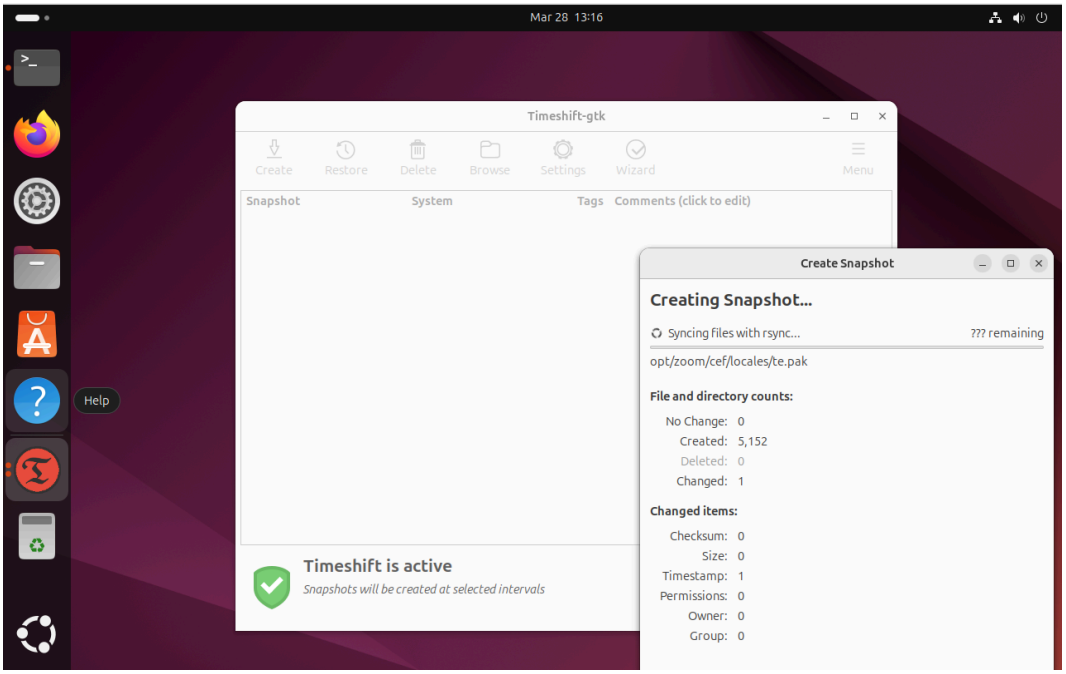


Figure. Timeshift creating snapshot.

Restore Workflows and Recovery Mindset

A backup strategy is incomplete until the user understands the restore. *Ubuntu: The Complete Guide* is very strong on this point. It explicitly shows that Déjà Dup allows the user to review a summary before restoring, and it highlights that restore can target a **new folder** instead of overwriting the original immediately. That is a very good desktop habit because it reduces panic and makes validation easier.

This leads to an important recovery mindset:

- restore calmly, not blindly
- preview what will be restored
- where possible, restore to a new folder first
- confirm the right backup set is selected
- treat restore as part of normal administration, not only as a last resort

A useful illustration is to think of recovery like unpacking archived exam papers. You do not throw them straight onto the current desk. You first open them in a safe space and verify that they are the correct set.

CLI example: create a safe restore target

```
mkdir -p ~/Restore-Test
ls ~
```

Example output:

```
Backups  Documents  Downloads  Restore-Test
```

CLI example: verify recovered files after restore

```
ls ~/Restore-Test
```

Example output:

```
Documents  Pictures  notes.txt
```

This simple example reflects the restore-first-to-a-safe-location mindset shown in *Ubuntu: The Complete Guide*.

A strong recovery mindset also includes one more important habit: **practice restore before a disaster.** A backup that has never been tested is only a hopeful assumption.

Screenshot notes:

[Screenshot-CH18-14: Déjà Dup restore summary screen showing source backup and restore destination.]
[Screenshot-CH18-15: Restored files appearing in a safe test folder instead of overwriting originals immediately.]

Guided Maintenance and Backup Lab

To make this chapter practical, readers can complete a short hands-on exercise:

- run `sudo apt update` and `sudo apt upgrade`
- inspect `apt list --upgradable`
- install `unattended-upgrades`
- check disk space with `df -h`
- run `sudo apt autoremove` and `sudo apt clean`
- install and open **Déjà Dup**
- configure a simple backup target
- create a safe restore folder such as `~/Restore-Test`
- install and open **Timeshift**
- create one snapshot before making another system change

This kind of exercise turns updates and backup from passive theory into a repeatable desktop routine.

Chapter Reflection

Updates, maintenance, and backup are the quiet disciplines that keep Ubuntu Desktop reliable over time. Manual and automatic updates work together to balance control and safety. Security fixes should be applied promptly, while broader package and release changes should be handled with a sensible patch strategy. `unattended-upgrades` helps automate urgent maintenance. Routine upkeep includes checking disk usage, reviewing package state, and cleaning stale cache or dependencies when appropriate. Déjà Dup provides approachable, incremental backups and practical restore workflows, especially when the user restores to a safe location first. Snapshot tools such as Timeshift complement file backups by preserving system state before riskier changes. Together, these habits create not only a healthier desktop, but also a more confident desktop user—one who expects problems to happen sometimes and prepares for recovery in advance.

Key Takeaways

- **Ubuntu Desktop supports both manual and automatic updates**, and both are useful.
- **Security updates should be applied quickly**, while broader changes should follow a sensible patch strategy.
- **`unattended-upgrades`** is the standard Ubuntu tool for automated urgent patching.
- **Routine maintenance includes cleanup**, not just updates.
- **Déjà Dup backups are incremental** and can be restored safely to alternate folders first.
- **Snapshots and file backups serve different roles**, and together they provide stronger recovery coverage.
- **A tested restore workflow is as important as the backup itself.**

Part IX — Troubleshooting Ubuntu Desktop

Chapter 19 — Solving Common Desktop Problems

Even a well-maintained Ubuntu desktop will sometimes misbehave. An application may freeze, audio may disappear, Wi-Fi may connect but not reach the internet, a graphics driver may break the session, or a GNOME extension may make the desktop unstable. The Ubuntu Academy outline explicitly places **frozen apps, driver conflicts, slow boot, cache cleanup, and troubleshooting common issues** inside everyday desktop operations and best practices, which makes this chapter a very practical part of the book.

The attached references support this chapter from different angles. *Ubuntu Linux Bible* provides practical package-repair and process-management tools such as `apt --fix-broken`, `dpkg-reconfigure`, `apt autoremove`, `ps`, `top`, and System Monitor. *Ubuntu: The Complete Guide* gives concrete troubleshooting paths for sound, Bluetooth, and boot issues, including tools such as **alsamixer**, **PulseAudio Volume Control**, and **Boot-Repair**. *Mastering Ubuntu* reinforces the value of Snap background updates and GNOME Shell customization awareness, both of which matter when debugging application behavior and extension-related conflicts.

A useful illustration is to think of desktop troubleshooting like working through a building problem. If the lights go out, the first question is not “replace the whole building.” It is: **which room is affected, what changed recently, and what does the control panel say?** Good Ubuntu troubleshooting follows the same logic: identify the scope, isolate the layer, and use the right tool for that layer.

Frozen Applications and Unresponsive Sessions

One of the most common desktop problems is a frozen application. A browser tab may hang, a media player may stop responding, or the entire session may feel sluggish. The Ubuntu Linux Bible strongly supports this section because it explicitly teaches **process management** through `ps`, `top`, and **System Monitor**, and explains that Linux desktops can identify and stop misbehaving processes without rebooting the whole machine.

A good first rule is: **distinguish between a frozen app and a frozen desktop session**. If one app is broken but the mouse, keyboard, or other apps still respond, the problem is usually smaller and easier to fix. If the entire graphical session is unresponsive, the problem may be deeper—graphics, memory pressure, GNOME Shell, or driver instability.

A practical response order is:

- try closing the app normally
- use **System Monitor** or `ps` to identify the process
- terminate or kill only the stuck app first
- reboot only if the session itself is unusable

A useful analogy is to think of a frozen application as one employee in an office refusing to move. You do not evacuate the whole building immediately. First, identify the one person causing the blockage.

CLI example: inspect heavy processes

```
ps -eo pid,user,group,gid,vsz,rss,comm --sort=-vsz | head
```

Example output:

```
  PID USER     GROUP      GID    VSZ   RSS COMMAND
 2366 student  student   1000 3720060 317060 gnome-shell
 3030 student  student   1000 2456968 248340 firefox
 3233 student  student   1000 2314388 316252 Web Content
```

The Ubuntu Linux Bible explicitly shows this kind of sorted ps output to identify large or heavy processes.

CLI example: live monitoring with top

```
top
```

Example output:

```
top - 10:02:11 up 2:25,  2 users,  load average: 0.48, 0.55, 0.52
Tasks: 286 total,   1 running, 285 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.4 us,  2.1 sy,  0.0 ni, 90.8 id,  0.3 wa,  0.0 hi,  0.4 si,  0.0 st
MiB Mem :   7800.0 total,   3890.0 free,   2110.0 used,   1800.0 buff/cache
```

CLI example: terminate a stuck application

```
kill 3030
```

Typical result:

```
(no output if successful)
```

CLI example: open the graphical System Monitor

```
gnome-system-monitor &
```

Example output:

```
[1] 6152
```

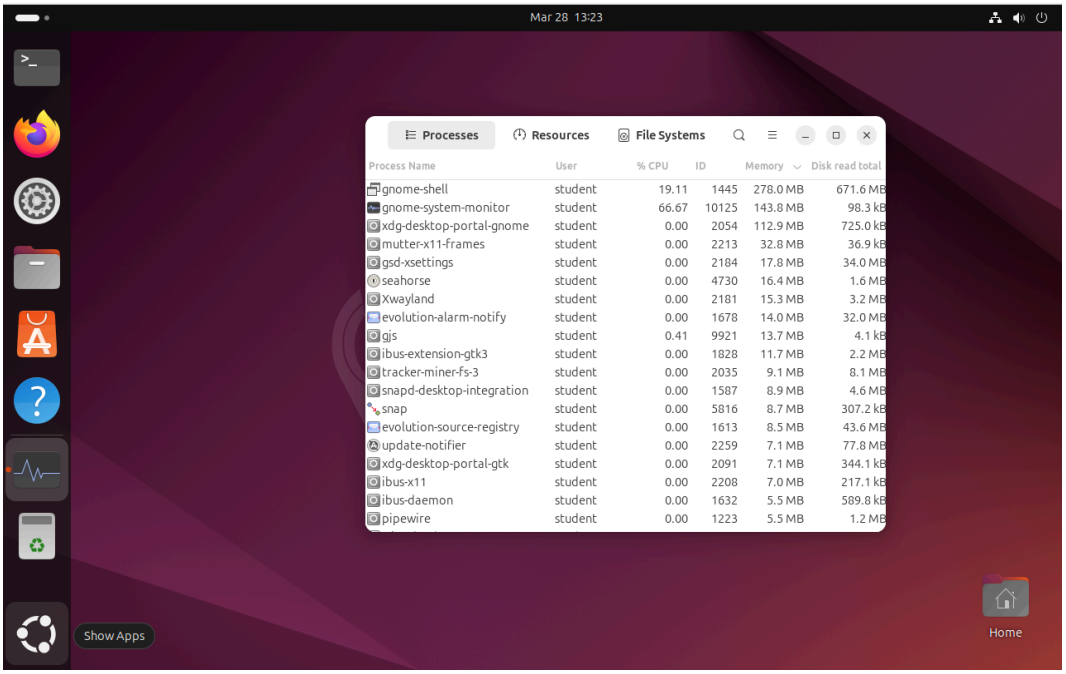


Figure. GNOME System Monitor sorted by CPU or Memory

Software Installation Problems

A large number of desktop problems are actually software-management problems. An application may fail to install, the package may not be found, or the installation may have been interrupted. The Ubuntu Linux Bible directly addresses this. It explains that interrupted installation can leave packages in an unusable state and that package management issues should often be repaired through APT and `dpkg` rather than by reinstalling the whole system.

The references also remind readers that Ubuntu's repository model is one of its strengths. If software comes from **official, supported repositories**, users are much less likely to run into trouble than when mixing random sources or unverified packages.

A practical installation-troubleshooting checklist is:

- confirm the package name is correct
- run `apt update`
- verify internet access
- check whether the repository containing the package is enabled
- repair broken dependencies if installation was interrupted
- reconfigure a misbehaving package if the problem is configuration rather than absence

A useful illustration is to think of package installation like loading books into a library catalog. If the catalog is outdated, the book may appear to "not exist." If the delivery was interrupted, the shelf may contain a half-filed copy.

CLI example: refresh package lists

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
```

CLI example: search for a package

```
apt-cache search vlc
```

Example output:

```
vlc - multimedia player and streamer
vlc-bin - binaries from VLC
```

CLI example: diagnose a package not found situation

```
sudo apt install package-that-does-not-exist
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package package-that-does-not-exist
```

The Ubuntu Linux Bible explicitly explains that “Unable to locate package” commonly means the local repo database is outdated or the package name was typed incorrectly.

CLI example: repair an interrupted installation state

```
sudo apt --fix-broken install
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Correcting dependencies... Done
```

This is directly supported by the Ubuntu Linux Bible’s package-repair guidance.

CLI example: reconfigure a misbehaving package

```
sudo dpkg-reconfigure package-name
```

Example output:

```
(package configuration dialog opens or package configuration is
reapplied)
```

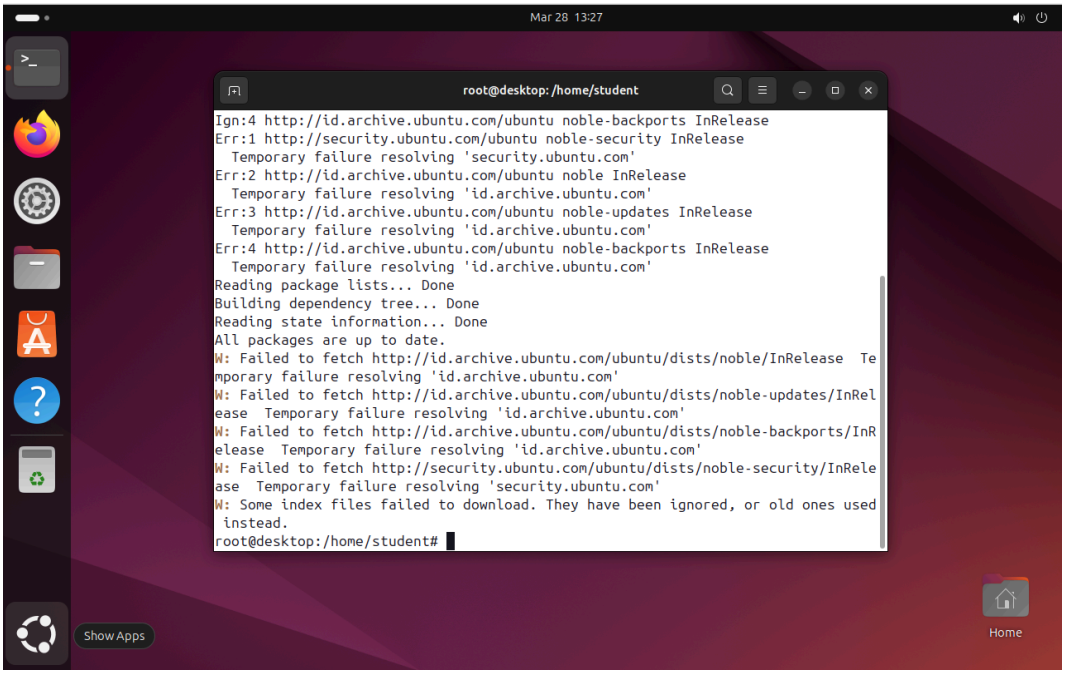


Figure. Terminal showing on “Temporary Failure”

Broken Dependencies and Repository Issues

Some of the most frustrating desktop problems come from **dependency breakage** or inconsistent repositories. The Ubuntu Linux Bible explicitly discusses this area. It notes that when installations are interrupted or repository conditions are inconsistent, the package manager can be left in a broken state. It recommends repair with `apt --fix-broken install`, package cleanup with `apt autoremove`, and safe treatment of repositories because unsupported or poorly maintained third-party sources increase the chance of trouble.

The same references also emphasize that official repositories are safer and more consistent, while PPAs or other third-party sources must be trusted carefully. A broken repository key or an outdated source file can cause update failures that look like “system problems” but are really **source-management problems**.

A practical dependency-and-repository troubleshooting path is:

- run `sudo apt update`
- repair broken installs with `sudo apt --fix-broken install`
- remove unused dependency packages with `sudo apt autoremove`
- inspect `/etc/apt/sources.list.d/`
- remove obsolete or untrusted repository files

A good illustration is to think of dependencies as stacked blocks. If one supporting block is damaged or missing, the visible application on top may collapse even if its own files look fine.

CLI example: repair broken dependencies

```
sudo apt --fix-broken install
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Correcting dependencies... Done
```

The following packages will be configured:

```
  example-runtime
```

CLI example: remove unneeded packages

```
sudo apt autoremove
```

Example output:

```
The following packages will be REMOVED:
  kde-cli-tools kde-cli-tools-data libfakekey0
After this operation, 5,385 kB disk space will be freed.
```

This mirrors the `apt autoremove` example in the Ubuntu Linux Bible.

CLI example: inspect repository source files

```
ls /etc/apt/sources.list.d/
```

Example output:

```
example-project-noble.list
official-extra.list
```

CLI example: remove an unwanted repository source and refresh

```
sudo rm
/etc/apt/sources.list.d/example-project-noble.list
sudo apt update
```

This closely follows the safe repository-cleanup logic already supported in the Ubuntu security and package-management references.

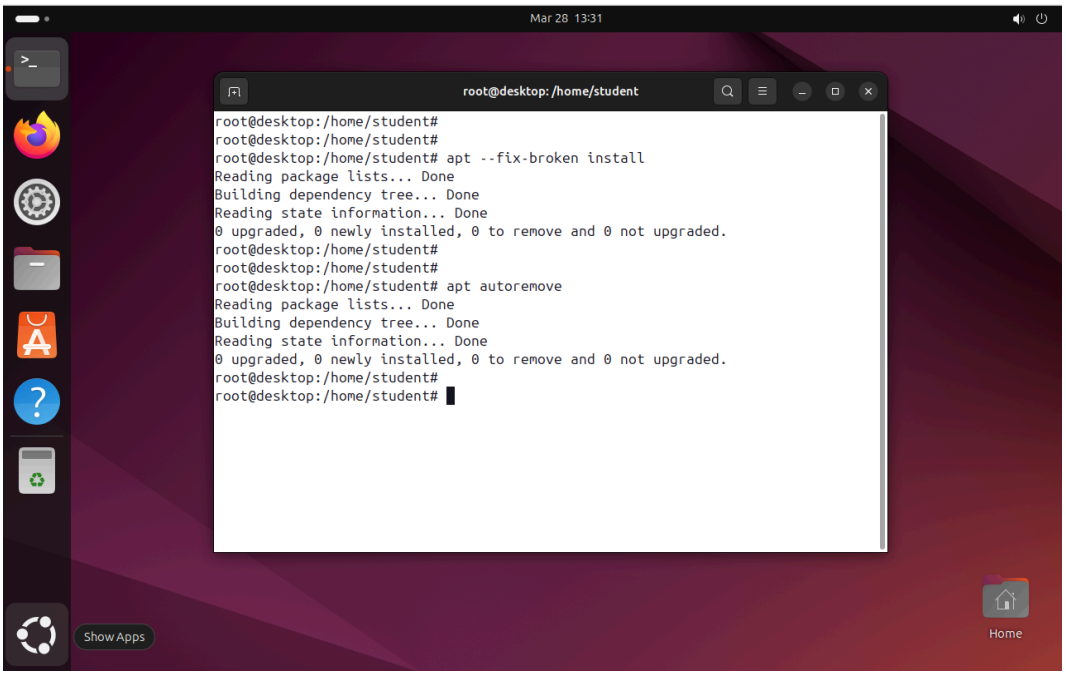


Figure. Terminal showing apt --fix-broken install followed by apt autoremove

Slow Startup and Performance Bottlenecks

Sometimes Ubuntu Desktop is not broken. It is just **slow**. Startup may take too long, login may feel heavy, or the desktop may respond sluggishly after boot. The Ubuntu Academy outline explicitly includes **slow boot** and **performance tuning**, which makes this one of the most important real-world chapters for ordinary users.

The attached references support this from several angles. The Ubuntu Linux Bible's process tools help identify CPU and memory pressure. The Ubuntu Complete Guide explains how to manage **Startup Applications** and even gives a concrete method using `gnome-session-properties`, which is extremely useful when too many login-time programs are slowing the session. Mastering Ubuntu also contrasts desktops in terms of perceived speed and responsiveness, which is a useful reminder that environment design and add-ons can affect startup feel.

A practical performance-troubleshooting strategy includes:

- check for heavy background processes
- inspect login-time startup items
- remove unnecessary extensions or extra desktop add-ons
- check free disk space
- clean caches and unused packages when appropriate
- distinguish system slowness from one single misbehaving application

A good analogy is to think of slow startup like arriving at a classroom where too many machines, lights, and apps power on at once. The room still works, but it takes longer to become comfortable.

CLI example: inspect uptime and load

```
uptime
```

Example output:

```
10:31:22 up 3:04,  2 users,  load average: 0.34, 0.41, 0.47
```

CLI example: check memory and disk pressure

```
free -h
df -h
```

Example output:

```
               total        used        free      shared  buff/cache   available
Mem:           7.6Gi       2.1Gi       3.8Gi       318Mi       1.7Gi       5.0Gi
Swap:          2.0Gi          0B       2.0Gi

Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p2  236G   34G  191G  15% /
```

CLI example: inspect startup applications

```
ls ~/.config/autostart
```

Example output:

```
telegram-desktop.desktop
update-notifier.desktop
```

CLI example: open Startup Applications graphically

```
gnome-session-properties &
```

Example output:

```
[1] 6280
```

The Ubuntu Complete Guide explicitly describes this tool and its role in controlling what launches at login.

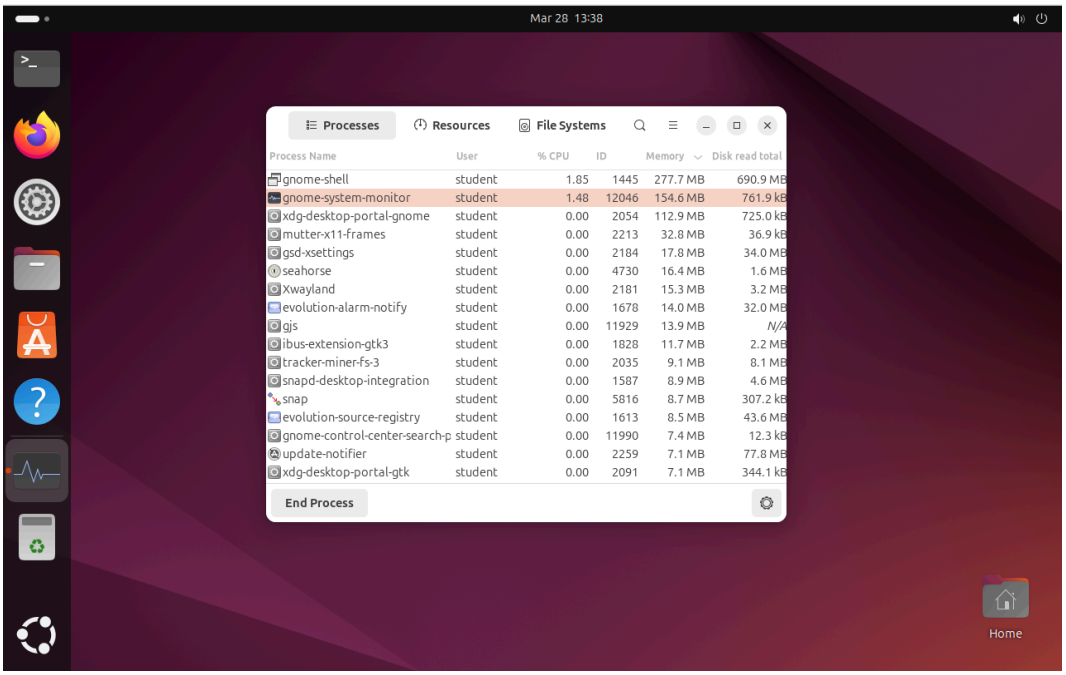


Figure. GNOME System Monitor highlighting one background process causing high CPU or RAM usage

Sound, Wi-Fi, Bluetooth, and Display Problems

Hardware-related desktop issues are some of the most visible and most frustrating. The attached references provide especially strong troubleshooting help here. *Ubuntu: The Complete Guide* walks through **sound troubleshooting**, including checking the correct output device, running `alsamixer`, testing playback with `aplay /usr/share/sounds/alsa/Front_Center.wav`, and resetting PulseAudio settings if necessary. It also gives a practical Bluetooth pairing sequence and notes that if no adapter is found, Ubuntu will report that directly. The Ubuntu Academy outline also treats sound, Bluetooth, Wi-Fi, and display setup as core desktop hardware topics.

A good multi-device troubleshooting order is:

- verify that Ubuntu detects the device
- confirm the correct output or connection is selected
- test with a simple, known-good method
- inspect logs or device listings if needed
- only then consider reinstalling drivers or resetting configuration

A useful analogy is to think of these problems like classroom equipment checks. First make sure the microphone exists, then make sure it is the one selected, then test it, and only then assume something deeper is broken.

CLI example: test sound output

```
aplay /usr/share/sounds/alsa/Front_Center.wav
```

Example output:

```
Playing WAVE '/usr/share/sounds/alsa/Front_Center.wav' : Signed
16 bit Little Endian, Rate 48000 Hz, Mono
```

The Complete Guide explicitly recommends this exact command for sound testing.

CLI example: open ALSA mixer

```
alsamixer
```

Example output:

```
(terminal opens interactive mixer with volume bars and output
devices)
```

CLI example: reset PulseAudio settings

```
sudo rm -r ~/.config/pulse; pulseaudio -k
```

Typical result:

```
(no output if successful)
```

This command is directly supported by Ubuntu: The Complete Guide.

CLI example: inspect Bluetooth adapter presence

```
lsusb | grep -i bluetooth
```

Example output:

```
Bus 001 Device 002: ID 8087:0a2b Intel Corp. Bluetooth wireless
interface
```

CLI example: inspect network connection state

```
nmcli device status
```

Example output:

```
DEVICE      TYPE      STATE         CONNECTION
wlp2s0      wifi      connected     CampusNet
lo          loopback  unmanaged     --
```

CLI example: inspect current display outputs

```
xrandr
```

Example output:

```
Screen 0: minimum 8 x 8, current 3840 x 1080, maximum 32767 x 32767
eDP-1 connected primary 1920x1080+0+0
HDMI-1 connected 1920x1080+1920+0
```

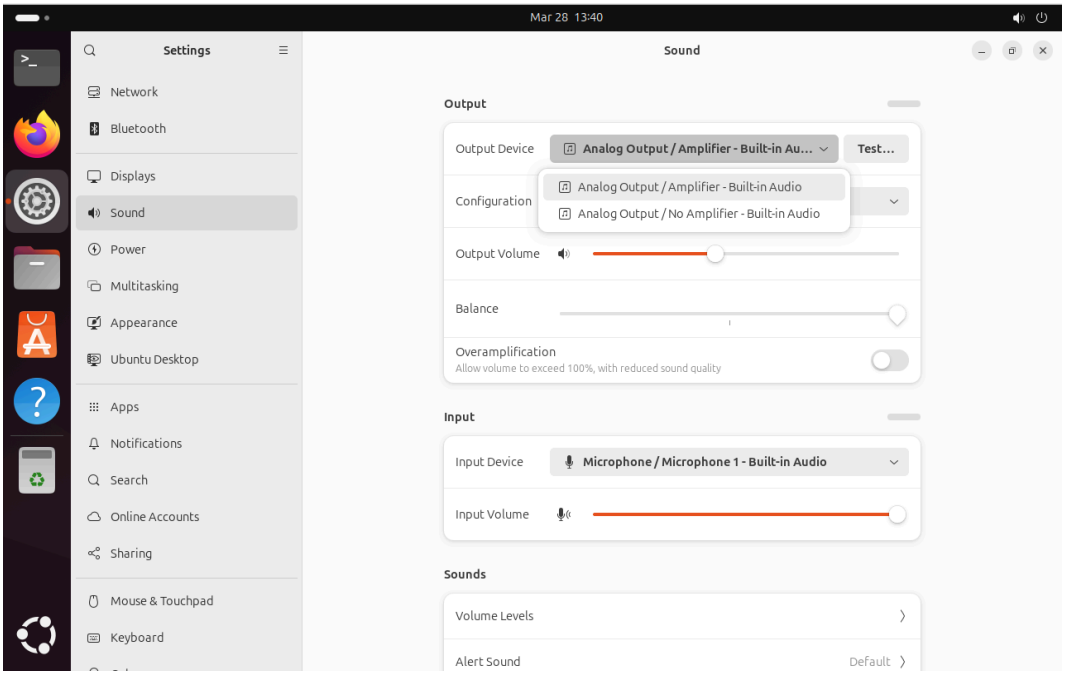


Figure. Settings > Sound showing the correct output device selected

Login Issues and User Session Problems

Login issues can be unsettling because they affect the user before the normal desktop even appears. Problems in this area may include wrong passwords, session loops, misconfigured startup items, or broader GNOME session problems. The references support this area in several ways. The Ubuntu Academy outline includes **user switching, guest sessions, password policies, startup applications, and session management**, while Ubuntu Complete Guide explicitly explains that startup applications and multi-user behavior are part of normal Ubuntu desktop administration.

A practical first response to login problems is:

- verify username and password carefully
- confirm keyboard layout and Caps Lock state
- try another local account if available
- inspect whether a specific user account is affected or all users are affected
- review startup application behavior if login succeeds but the session immediately becomes unstable

A good analogy is to think of login issues like entering a building with a badge. First confirm you are at the correct door, then confirm the badge is valid, then check whether the problem is your badge or the whole door system.

CLI example: confirm the current user after successful login

```
whoami
```

Example output:

```
student
```

CLI example: list active sessions

```
loginctl list-sessions
```

Example output:

```
SESSION  UID USER     SEAT  TTY
     2 1000 student  seat0 tty2
     3 1002 student2 seat0 tty3
```

CLI example: inspect user autostart items that may destabilize the session

```
ls ~/.config/autostart
```

Example output:

```
problematic-extension-helper.desktop
update-notifier.desktop
```

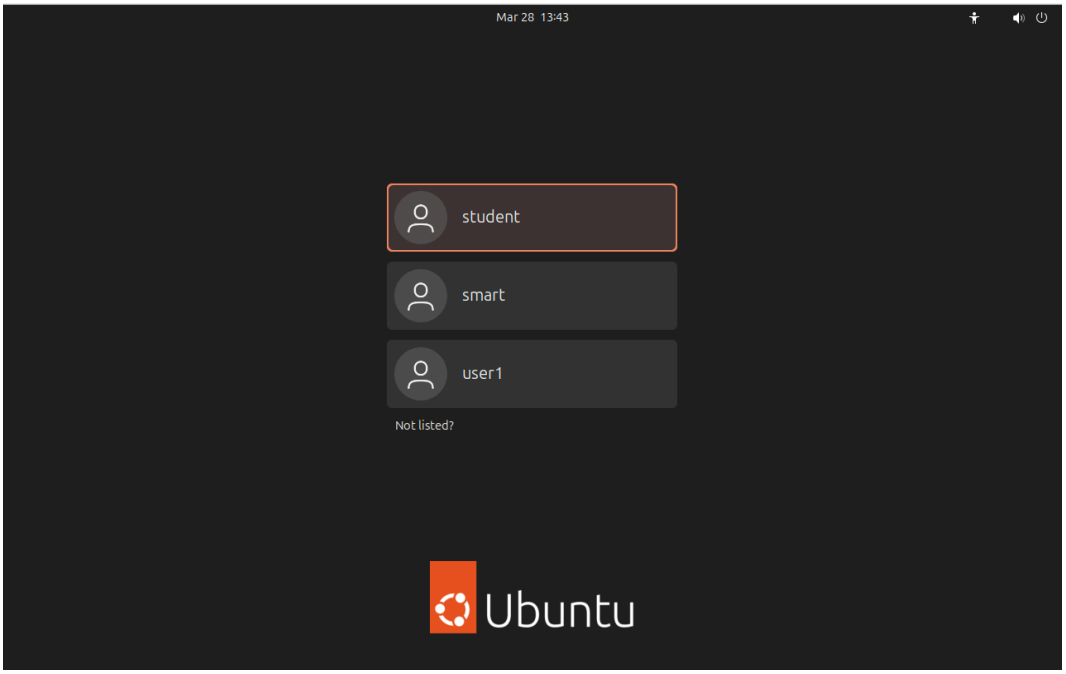


Figure. Ubuntu login screen showing multiple accounts available

Graphics Driver Conflicts

Graphics driver problems can cause some of the most dramatic desktop failures: black screens, login loops, missing acceleration, broken external displays, or GNOME instability. The Ubuntu Academy outline explicitly flags **driver conflicts**, **proprietary vs open drivers**, and **PRIME for hybrid graphics**, making this a major desktop-troubleshooting topic.

The attached references already established the key ideas: Ubuntu can identify graphics hardware with `lspci`, show recommended drivers through `ubuntu-drivers devices`, and switch behavior on hybrid systems through PRIME-related workflows. They also imply an important troubleshooting principle: **not every graphics problem needs a brand-new driver; sometimes the wrong driver is already installed**.

A practical graphics-conflict troubleshooting order is:

- identify the GPU model correctly
- inspect which driver is currently in use
- review available recommended drivers
- change only one graphics variable at a time
- keep external displays and hybrid graphics in mind

A good illustration is to think of a graphics driver like the translator between the desktop and the display hardware. If the wrong translator is speaking, the whole visual conversation becomes confused.

CLI example: identify graphics hardware

```
lspci | grep -i vga
```

Example output:

```
00:02.0 VGA compatible controller: Intel Corporation UHD Graphics 620
01:00.0 VGA compatible controller: NVIDIA Corporation GP108M [GeForce MX150]
```

CLI example: inspect available drivers

```
sudo ubuntu-drivers devices
```

Example output:

```
== /sys/devices/pci0000:00/0000:01:00.0 ==
vendor   : NVIDIA Corporation
model    : GP108M [GeForce MX150]
driver   : nvidia-driver-535 - third-party non-free recommended
driver   : xserver-xorg-video-nouveau - distro free builtin
```

CLI example: inspect current PRIME profile on a hybrid laptop

```
sudo prime-select query
```

Example output:

```
on-demand
```

CLI example: verify NVIDIA driver health

```
nvidia-smi
```

Example output:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.171.04    Driver Version: 535.171.04    CUDA Version: 12.2   |
+-----------------------------------------------------------------------------+
```

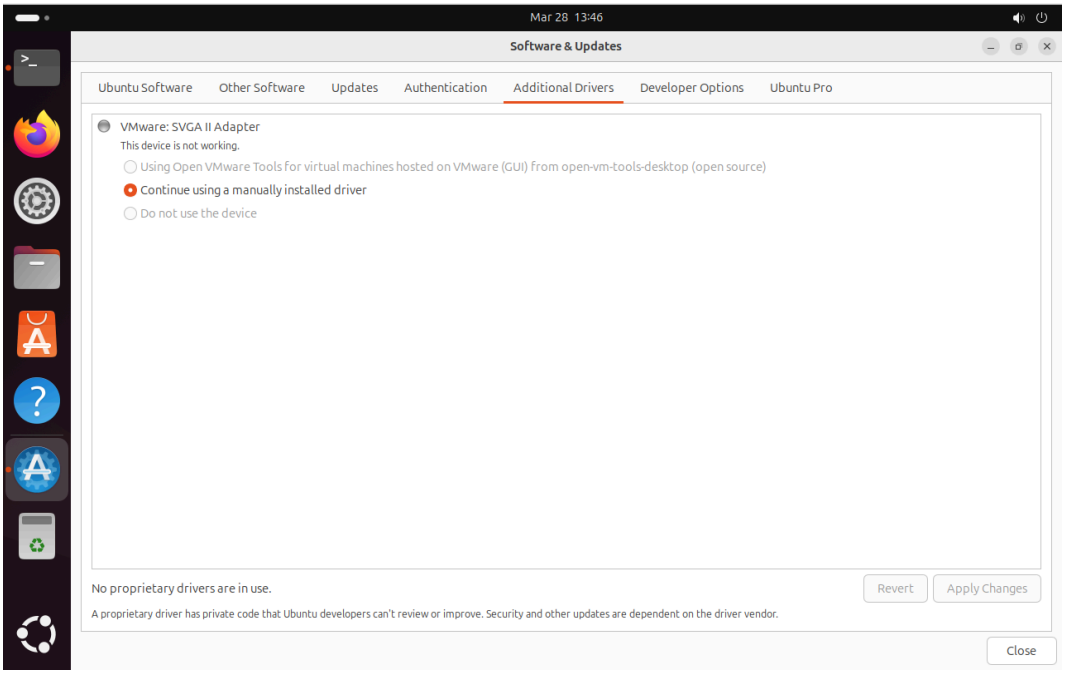


Figure. Additional Drivers tab showing open

App Crashes and Extension Conflicts

Sometimes the desktop environment itself is stable, but one application crashes repeatedly—or GNOME behaves strangely because an extension is conflicting with the current shell version. The Ubuntu Academy outline explicitly mentions **GNOME Extensions** and also highlights **common issues, frozen apps, and performance tuning**, which strongly supports including extension conflicts here.

The attached references also help. *Mastering Ubuntu* explicitly discusses **GNOME shell extensions**, and the Ubuntu customization material from earlier chapters shows that extensions modify shell behavior. This is useful, but it also means that after GNOME updates or theme changes, an extension may become unstable, out of date, or incompatible.

A strong troubleshooting habit is:

- suspect recently added extensions first when GNOME Shell becomes strange
- disable one extension at a time
- test the desktop between changes
- separate “the app crashes” from “the shell crashes”
- update or remove stale packages instead of endlessly retrying them

A good illustration is to think of extensions like small attachments added to a machine. They can make the machine better, but any one of them can also create vibration or instability if it does not fit properly.

CLI example: list enabled GNOME extensions

```
gnome-extensions list --enabled
```

Example output:

```
ubuntu-dock@ubuntu.com
ding@rastersoft.com
user-theme@gnome-shell-extensions.gcampax.github.com
```

CLI example: disable one extension for testing

```
gnome-extensions disable
user-theme@gnome-shell-extensions.gcampax.github.com
```

Typical result:

```
(no output if successful)
```

CLI example: inspect all installed extensions

```
gnome-extensions list
```

Example output:

```
ubuntu-dock@ubuntu.com
ding@rastersoft.com
user-theme@gnome-shell-extensions.gcampax.github.com
```

CLI example: reconfigure a recently broken package

```
sudo dpkg-reconfigure package-name
```

This is especially useful when an application is installed correctly but behaving badly because of misconfiguration, as the Ubuntu Linux Bible notes.

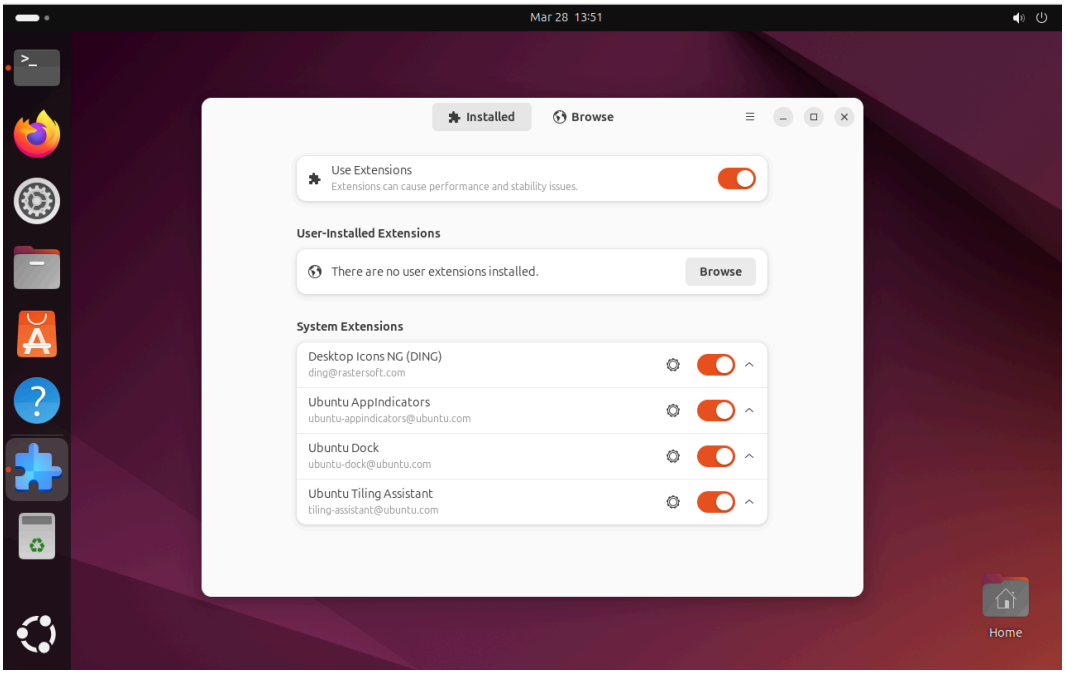


Figure. GNOME Extensions tool

Guided Desktop Troubleshooting Lab

To make this chapter practical, readers can complete a short troubleshooting exercise:

- open **System Monitor** and identify one heavy process
- run `top` and `ps` to compare process visibility
- simulate a package issue and repair it with `apt --fix-broken install`
- run `apt autoremove`
- test sound with `aplay /usr/share/sounds/alsa/Front_Center.wav`
- inspect Bluetooth with `lsusb | grep -i bluetooth`
- inspect Wi-Fi state with `nmcli device status`
- list display outputs with `xrandr`
- inspect current graphics driver suggestions with `ubuntu-drivers devices`
- list enabled GNOME extensions and disable one temporarily for testing

This kind of exercise teaches a vital lesson: **troubleshooting is not guessing. It is controlled observation followed by small, reversible actions**.

Chapter Reflection

Solving common desktop problems on Ubuntu requires a calm, layered approach. Frozen applications and sluggish sessions are often best handled through process tools such as `ps`, `top`, and System Monitor. Software installation failures frequently come down to package metadata, interrupted installs, broken dependencies, or repository issues, which can often be repaired through APT and `dpkg` rather than drastic measures. Slow startup and general performance problems often involve startup items, excessive background activity, or unnecessary clutter. Sound, Wi-Fi, Bluetooth, and display issues should be approached through detection, device selection, and simple known-good tests before attempting bigger changes. Login and session problems benefit from distinguishing account-specific issues from system-wide ones. Graphics driver conflicts require especially careful, one-variable-at-a-time troubleshooting. Finally, GNOME extensions and application configuration must be treated as possible causes when crashes or strange shell behavior appear after customization. Ubuntu Desktop becomes much less intimidating once users realize that most “big” failures can be broken into smaller, understandable layers.

Key Takeaways

- **Most desktop problems become easier once you identify the affected layer**: app, session, package, driver, or hardware.
- **`ps`, `top`, and System Monitor** are essential for frozen apps and heavy-process troubleshooting.
- **`apt --fix-broken install`, `dpkg-reconfigure`, and `apt autoremove`** are core package-repair tools.
- **Sound and Bluetooth issues should be tested with simple known-good tools first**, such as `aplay` and `alsamixer`.
- **Graphics problems often come from driver mismatch or hybrid-GPU configuration**, not from the desktop environment alone.
- **GNOME extensions can improve the desktop, but they can also destabilize it** after changes or updates.
- **Troubleshooting works best when changes are small, observable, and reversible.**

Chapter 20 — Boot and Recovery Troubleshooting

Boot failures are some of the most stressful problems a desktop user can face. When Ubuntu no longer reaches the login screen, many users assume the whole system is lost. In reality, most boot problems can be understood more calmly when the process is divided into stages and the right recovery tools are used. The Ubuntu Academy outline explicitly includes **GRUB2 recovery, initramfs problems, the Boot-Repair tool, live USB rescue, and saving user files before repair**, which makes this chapter a core part of practical Ubuntu Desktop administration.

The attached references support this chapter strongly. *Ubuntu: The Complete Guide* breaks startup problems into **pre-GRUB, GRUB, and post-GRUB** stages, explains that the simplest fix for many GRUB failures is often **Boot-Repair**, and recommends using rescue media to check disks and back up important files before deeper repair attempts. *Ubuntu Linux Bible* explains GRUB boot interruption, kernel boot options, the role of `initrd`, and the use of live media as a practical recovery path.

A useful illustration is to think of the boot process like opening a building for the day. **Firmware and disk detection** are the main gates opening. **GRUB** is the reception desk that decides which route to take. **Kernel and initramfs** are the staff unlocking the inner rooms. The **desktop session** is the building becoming ready for work. When something fails, the goal is not to panic. The goal is to identify **which door did not open**.

Understanding the Boot Process at a Desktop Level

A desktop user does not need to become a firmware engineer to troubleshoot boot problems effectively, but they do need a simple model of how Ubuntu starts. *Ubuntu: The Complete Guide* gives an especially useful framework by dividing startup issues into three broad stages centered around **GRUB 2**: **pre-GRUB**, **GRUB**, and **post-GRUB**. This is an excellent mental model because it immediately narrows the problem space.

A practical desktop-level boot sequence is:

- the machine powers on and firmware checks hardware
- the boot device is selected
- **GRUB** loads and presents or follows a boot entry
- the Linux kernel and **initrd/initramfs** load
- the root filesystem becomes available
- system services start
- the graphical login or desktop session appears

This matters because different symptoms point to different stages:

- **nothing at all** may suggest power, firmware, or hardware issues
- **GRUB appears but Ubuntu does not continue** often points to GRUB config or post-GRUB issues

- **Ubuntu starts loading then freezes** often indicates kernel, initramfs, driver, or filesystem trouble

A helpful analogy is to imagine an airport. Firmware is the security gate, GRUB is the departure board, the kernel is the plane taking off, and the desktop session is the plane reaching cruising altitude. If the process fails, the first question is: **at which stage did travel stop?**

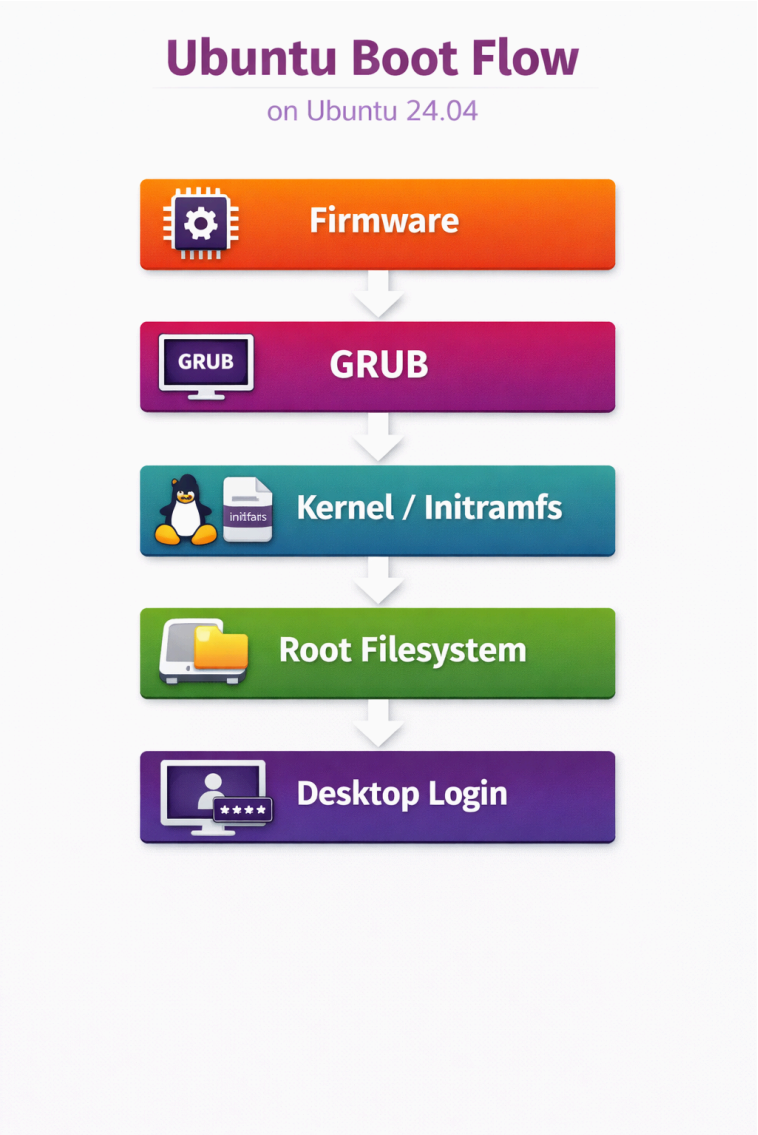


Figure. Diagram of the Ubuntu boot flow

GRUB Basics

GRUB is the boot loader that helps Ubuntu decide how to start. The Ubuntu Linux Bible explicitly identifies GRUB as a key part of the Linux installation and boot process, and *Ubuntu: The Complete Guide* explains that if the GRUB menu appears but Ubuntu does not continue properly, then the issue may lie in GRUB configuration or later stages of startup.

The Ubuntu Linux Bible also explains that users can force the GRUB menu to appear by pressing **Shift** early in the boot process, and that pressing **E** on a selected entry allows temporary editing of boot parameters. It notes that the kernel line includes the compressed kernel and the **initrd.img**, and that additional options can be added at the end of that line for troubleshooting.

At a desktop level, readers should understand that GRUB is responsible for:

- locating the boot entry
- passing parameters to the kernel
- handing off to the Linux startup process
- offering manual options when normal startup fails

A useful illustration is to think of GRUB as the receptionist at the entrance to the operating system. It does not do the actual desktop work, but it decides **which path** the system takes.

CLI example: inspect the current boot entry source file

```
ls /boot/grub
```

Example output:

```
fonts  grub.cfg  locale  themes  x86_64-efi
```

CLI example: show installed kernel images

```
ls /boot | grep vmlinuz
```

Example output:

```
vmlinuz
vmlinuz-6.8.0-52-generic
```

CLI example: show initramfs images

```
ls /boot | grep initrd
```

Example output:

```
initrd.img
initrd.img-6.8.0-52-generic
```

This matches the Ubuntu Linux Bible's explanation that GRUB passes both the kernel image and the initial RAM disk to the boot process.

CLI example: update GRUB configuration after a repair

```
sudo update-grub
```

Example output:

```
Sourcing file `/etc/default/grub'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.8.0-52-generic
Found initrd image: /boot/initrd.img-6.8.0-52-generic
done
```

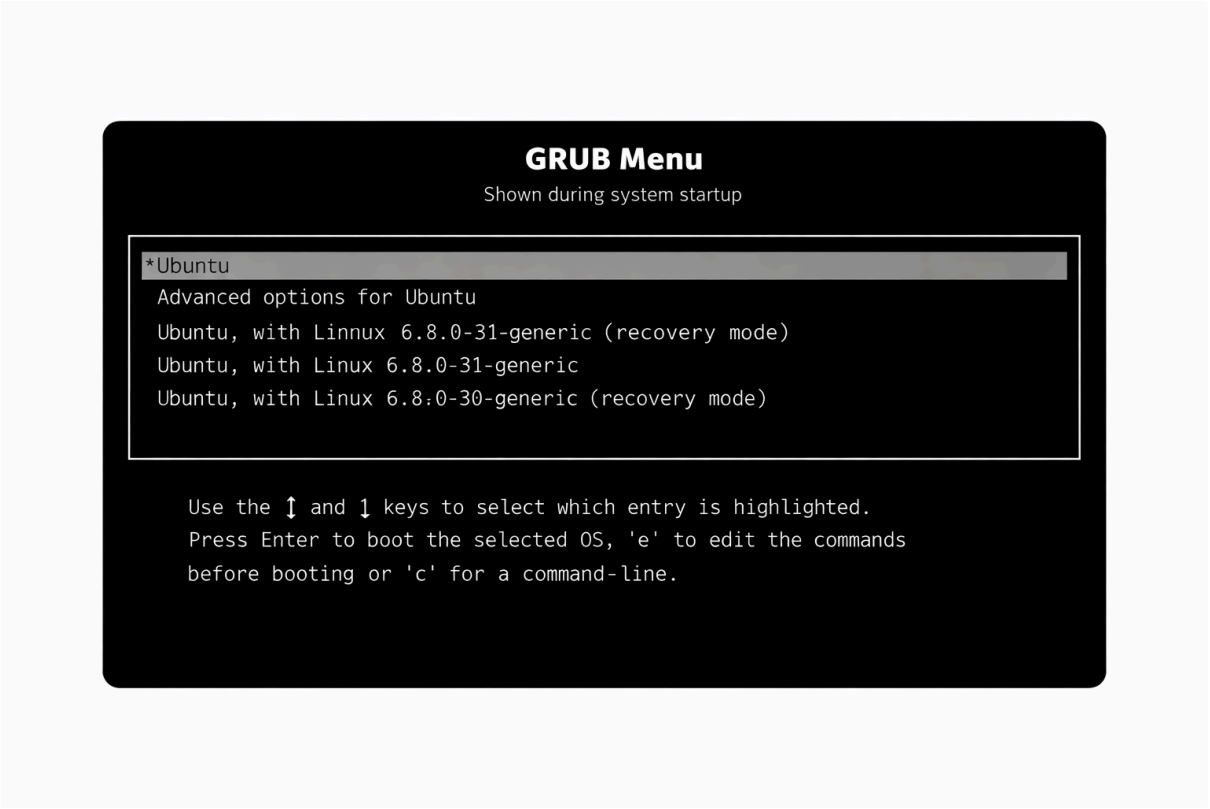


Figure. GRUB menu visible during startup with Ubuntu entries listed

Common Boot Failures

Boot failures usually feel chaotic, but *Ubuntu: The Complete Guide* makes an important point: many startup issues can be categorized by where in the boot process they occur. It describes cases such as a **blank screen**, a **frozen splash screen**, GRUB appearing but not progressing, or Linux starting to load and then halting. It recommends using those symptoms to decide where to focus troubleshooting.

A practical desktop list of common boot failures includes:

- the system powers on but no boot media is reached
- GRUB does not appear at all
- GRUB appears but booting a menu entry fails immediately
- the kernel starts but stalls during loading
- a graphics-related black screen appears after boot seems to continue
- a driver or filesystem issue prevents the session from finishing startup

The Ubuntu Linux Bible adds useful detail by explaining that boot failures can sometimes be worked around with **boot options** typed into the GRUB kernel line. It gives examples such as `acpi=off`, `noipv6`, and video-related options like `xdriver=vesa` or an explicit `resolution=1024×768` when display issues interfere with startup.

A helpful troubleshooting mindset is:

- if the machine does not even reach GRUB, think **hardware, firmware, disk selection, or boot sector**
- if GRUB appears but selection fails, think **GRUB configuration**
- if Linux begins to load and then fails, think **kernel, initramfs, drivers, or filesystem**

A good analogy is to think of boot failure like a train route. If the train never leaves the station, the issue is different from one that leaves the station but stops between cities.

CLI example: inspect the current kernel version on a working system

```
uname -r
```

Example output:

```
6.8.0-52-generic
```

CLI example: view recent current-boot logs on a system that did boot

```
journalctl -b -n 20
```

Example output:

```
Mar 14 07:37:10 ubuntu-desktop kernel: Linux version 6.8.0-52-generic ...
Mar 14 07:37:15 ubuntu-desktop systemd[1]: Started GNOME Display Manager.
Mar 14 07:37:30 ubuntu-desktop gdm-password][1014]:
pam_unix(gdm-password:session): session opened
```

This is useful because even successful boots help readers understand what a normal boot looks like before diagnosing an abnormal one.

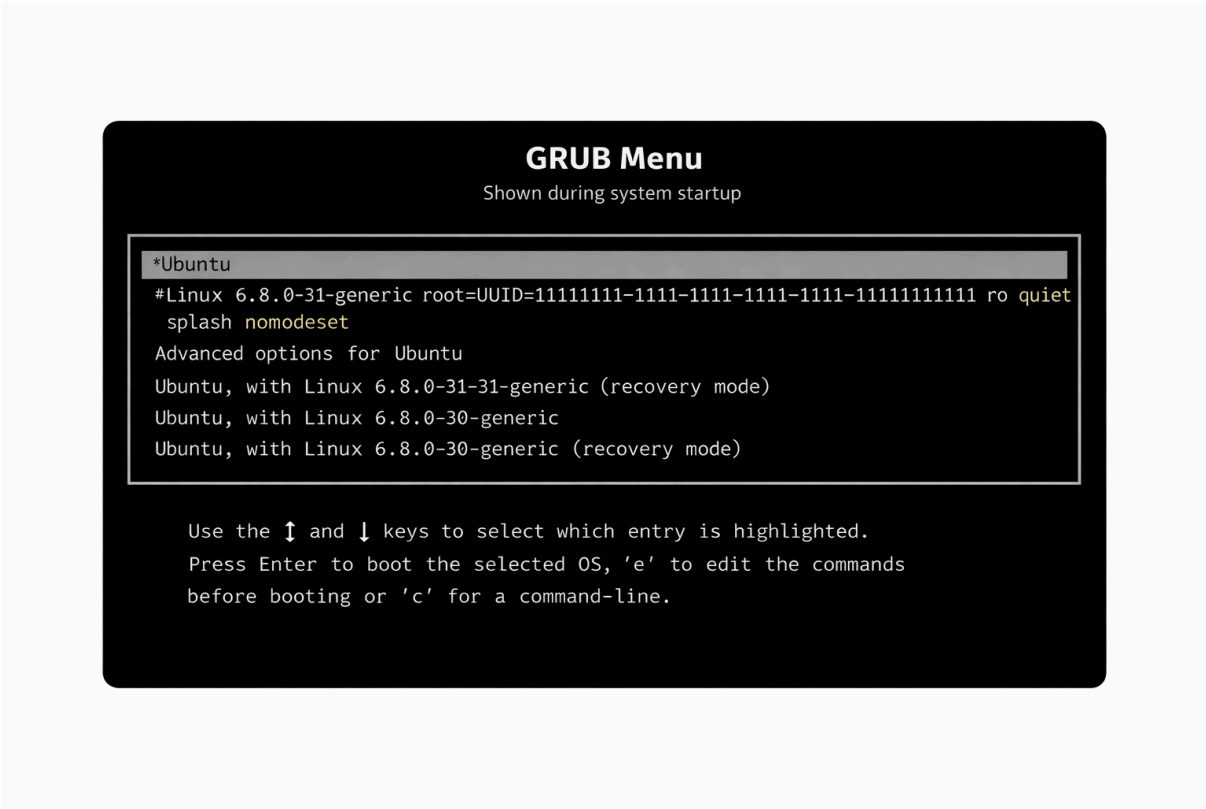


Figure. GRUB menu entry edited to add a temporary troubleshooting boot parameter

`initramfs` and Recovery Mode Basics

The Ubuntu Academy outline explicitly includes **initramfs problems** and recovery workflows, and the Ubuntu Linux Bible gives the essential conceptual foundation. It explains that the `initrd.img` on the GRUB kernel line is the **initial RAM disk**, containing modules and tools needed to start the installer or early boot environment. That means `initramfs` problems usually happen in the stage where the kernel is trying to reach the real root filesystem and load the modules needed to continue.

At a desktop level, the key point is simple: **initramfs is part of the bridge between GRUB and the full Ubuntu system**. If that bridge is broken, the system may stop with a shell prompt or fail before the desktop login appears.

Recovery Mode is useful because it offers a safer, reduced startup path for repair tasks. Even when the attached excerpts do not give a long procedural breakdown of Ubuntu's recovery menu itself, the Ubuntu Academy outline places recovery alongside boot repair and initramfs troubleshooting, which supports its inclusion here as a practical boot-safety concept.

A useful illustration is to think of initramfs as the emergency toolkit the system carries with it before the full workshop opens. If the workshop door will not open, that small toolkit may still let you investigate.

CLI example: rebuild the initramfs for the current kernel

```
sudo update-initramfs -u
```

Example output:

```
update-initramfs: Generating /boot/initrd.img-6.8.0-52-generic
```

CLI example: rebuild initramfs for all installed kernels

```
sudo update-initramfs -u -k all
```

Example output:

```
update-initramfs: Generating /boot/initrd.img-6.8.0-52-generic
update-initramfs: Generating /boot/initrd.img-6.8.0-47-generic
```

CLI example: confirm the initramfs files exist

```
ls /boot | grep initrd
```

Example output:

```
initrd.img
initrd.img-6.8.0-47-generic
initrd.img-6.8.0-52-generic
```

These are practical commands for post-repair work after booting from recovery mode or a live environment.

Screenshot notes:

`[Screenshot-CH20-07: GRUB advanced options menu showing a recovery mode entry.]`

[Screenshot-CH20-08: initramfs or recovery shell prompt with explanation of what stage failed.]

Boot-Repair Tool

Ubuntu: The Complete Guide is especially strong here. It clearly states that when GRUB issues are difficult to fix by hand, the **simplest fix is often Boot-Repair**. It explains that the tool is designed for **early boot problems caused by the boot sector, MBR, and GRUB**, and that its **Recommended repair** option fixes many common cases without requiring the user to work directly with low-level tools. It also notes that Boot-Repair generates logs that can be shared on forums for further help.

This is a perfect example of desktop-focused recovery: users do not always need to become GRUB experts. Sometimes the most professional answer is to use a well-designed recovery tool first.

The same reference gives a live-environment installation flow for Boot-Repair:

- add the Boot-Repair PPA
- update APT metadata
- install the tool
- launch it

A helpful illustration is to think of Boot-Repair as a trained repair technician arriving with a diagnostic checklist and the most common replacement parts already prepared.

CLI example: install Boot-Repair in a live environment

```
sudo add-apt-repository ppa:yannubuntu/boot-repair
sudo apt-get update
sudo apt-get install boot-repair
boot-repair
```

Example output:

```
Repository: 'deb
https://ppa.launchpadcontent.net/yannubuntu/boot-repair/ubuntu
noble main'
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Reading package lists... Done
Setting up boot-repair ...
```

This command sequence comes directly from *Ubuntu: The Complete Guide*.

A practical recommendation for readers is:

- try **Recommended repair** first
- save the generated log
- only move into advanced options when there is a clear reason

Screenshot notes:

[Screenshot-CH20-09: Boot-Repair main window showing the “Recommended repair” button.]

[Screenshot-CH20-10: Boot-Repair advanced options screen with contextual settings.]

Live USB Rescue Workflows

The Ubuntu Linux Bible and *Ubuntu: The Complete Guide* both strongly support using **live media** as a rescue environment. The Bible describes live installation media as a valid way to boot a Linux environment, and *Ubuntu: The Complete Guide* explicitly says that when boot problems occur, the rescue disc or live USB is invaluable for checking whether partitions are visible, opening the file manager, and accessing user files before attempting repairs.

This is one of the most important desktop recovery lessons in the entire chapter: **a broken boot does not automatically mean lost data**.

A typical live USB rescue workflow is:

- boot from the Ubuntu live USB
- choose the live session instead of immediate installation
- open **Files** or **Disks**
- locate the installed system's partitions
- back up important user files
- then proceed to repair attempts such as Boot-Repair or manual GRUB work

A good analogy is to think of the live USB as a temporary emergency office built outside the damaged building. You can still enter the storage rooms and retrieve important documents even if the main entrance is blocked.

CLI example: identify disks and partitions from a live session

```
lsblk -f
```

Example output:

```
NAME        FSTYPE      LABEL   UUID                   MOUNTPOINTS
sda
├─sda1      vfat        EFI     1234-ABCD
└─sda2      ext4                a1b2c3d4-e5f6-7788-99a
sdb         iso9660     Ubuntu  2026-03-14-08-00-00-00 /cdrom
```

CLI example: mount the installed system partition in the live session

```
sudo mkdir -p /mnt/rescue
sudo mount /dev/sda2 /mnt/rescue
ls /mnt/rescue
```

Example output:

```
bin  boot  home  etc  usr  var
```

CLI example: launch the Disks utility from the live session

```
gnome-disks &
```

Example output:

```
[1] 7812
```

Ubuntu: The Complete Guide explicitly recommends using the **Disks** utility from a live environment to verify whether drives and partitions are visible when boot repair is needed.

Screenshot notes:

[Screenshot-CH20-11: Ubuntu live session desktop running from USB, with Files and Disks available for rescue work.]
[Screenshot-CH20-12: Disks utility showing the installed system disk and its partitions from the live environment.]

Saving User Files Before Repair

One of the most practical warnings in *Ubuntu: The Complete Guide* is that when using rescue media, users should take the opportunity to **back up any precious files before proceeding further**. It states this very clearly in the context of non-GRUB boot issues: after booting rescue media, open the file manager and see whether the partitions are visible and accessible, because this is the right time to save important data.

This is essential desktop-recovery advice. Before risky repairs, always ask:

- are the user's files accessible?
- are they backed up already?
- can they be copied to an external USB drive or network location?
- is the system repair worth doing before protecting the data?

A good illustration is to think of file rescue like removing important notebooks from a classroom before repairing a leaking roof.

CLI example: copy a user's Documents folder to an external drive

```
sudo cp -a /mnt/rescue/home/student/Documents
/media/ubuntu/USB32/
```

Typical result:

```
(no output if successful)
```

CLI example: verify that the copied files exist

```
ls /media/ubuntu/USB32/Documents
```

Example output:

```
report.docx  presentation.odp  project-notes.txt
```

CLI example: copy user data to a second internal or network target

```
sudo rsync -avh /mnt/rescue/home/student/
/media/ubuntu/USB32/student-backup/
```

Example output:

```
sending incremental file list
Documents/report.docx
Pictures/wallpaper.jpg
```

These examples teach a very valuable principle: **repair comes after preservation**.

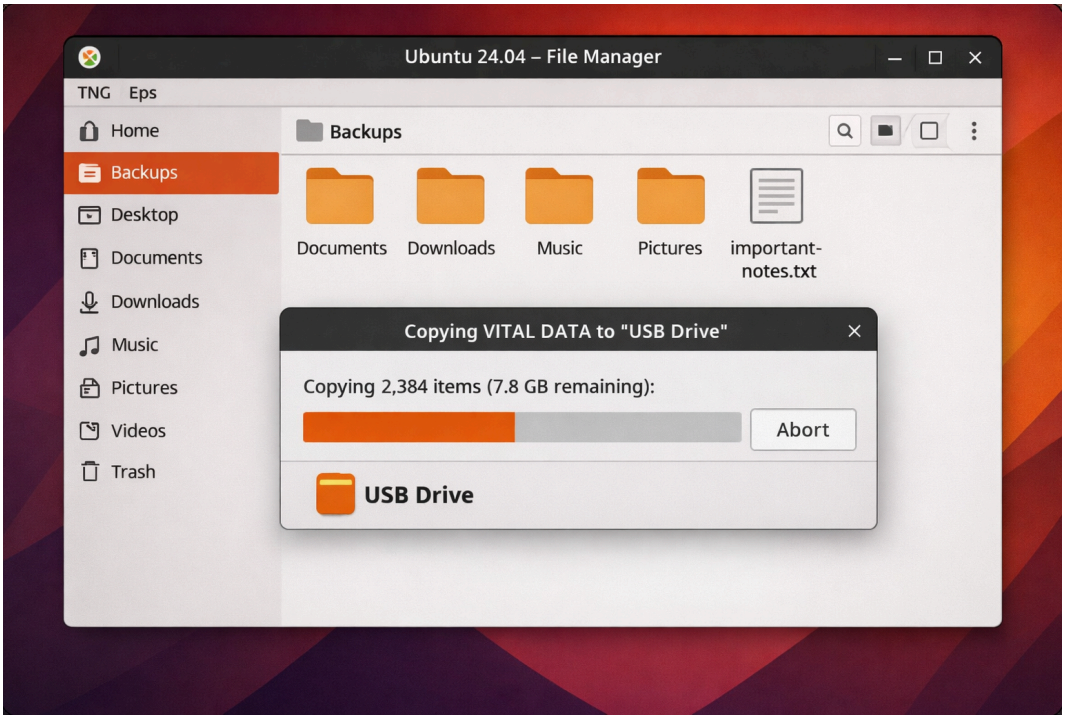


Figure. External USB drive connected and receiving copied backup files before repair attempts

Practical Desktop Recovery Checklist

A good desktop recovery workflow is easier to remember when turned into a checklist. The Ubuntu Academy outline explicitly frames this chapter as practical troubleshooting, and the references strongly support that mindset with live USB rescue, Boot-Repair, GRUB editing, and backup-first advice.

A practical Ubuntu Desktop recovery checklist is:

- identify **which stage** the boot failure belongs to
  - pre-GRUB
  - GRUB
  - post-GRUB
- try showing the **GRUB menu** and checking available entries
- if needed, use temporary **boot options** from GRUB to work around hardware or graphics problems
- boot a **live USB session**
- verify that the installed system's partitions are visible in **Files** or **Disks**
- **save user files before risky repairs**
- use **Boot-Repair** for early bootloader issues
- rebuild **GRUB** or **initramfs** if a manual repair path is needed
- only then attempt deeper recovery changes

A useful illustration is to compare this checklist to emergency response in a building: identify the blocked entrance, secure the people and valuables, use the standard rescue tools, and only then move into deeper reconstruction.

CLI example: quick recovery toolkit summary from a live session

```
lsblk -f
sudo mount /dev/sda2 /mnt/rescue
ls /mnt/rescue/home
sudo update-initramfs -u
sudo update-grub
```

Example output:

```
NAME        FSTYPE      LABEL   UUID                    MOUNTPOINTS
sda
├─sda1      vfat        EFI     1234-ABCD
└─sda2      ext4                a1b2c3d4-e5f6-7788-99aa
student
update-initramfs: Generating /boot/initrd.img-6.8.0-52-generic
Generating grub configuration file ...
done
```

This sequence is not the answer to every boot problem, but it captures the spirit of **detect, preserve, repair** very well.



Figure. Recovery checklist graphic with stages

Guided Boot Recovery Lab

To make this chapter practical, readers can complete a safe recovery-oriented exercise in a VM or lab machine:

- force the **GRUB menu** to appear during boot
- inspect the kernel and `initrd` lines with **E**
- boot a **live Ubuntu USB session**
- run `lsblk -f` and identify the installed system partition
- mount the partition under `/mnt/rescue`
- copy one user folder to a separate backup target
- install and launch **Boot-Repair** from the live environment
- run `update-initramfs -u` and `update-grub` in an appropriate repair context
- reboot and confirm whether the normal desktop login is restored

This kind of lab helps learners replace panic with procedure.

Chapter Reflection

Boot and recovery troubleshooting on Ubuntu Desktop becomes far less intimidating once the boot process is understood in stages. The most useful conceptual split is **pre-GRUB, GRUB, and post-GRUB**, because it narrows where the failure likely lives. GRUB serves as the boot handoff point and can often be inspected or temporarily adjusted directly from the boot menu. Common failures range from missing boot menus and black screens to GRUB configuration problems and post-GRUB freezes. `initramfs` and recovery mode matter because they bridge the kernel to the full operating system. For many early boot problems, **Boot-Repair** is the simplest and most practical graphical recovery tool. Live USB sessions provide an emergency environment for inspection, disk access, and rescue work. Most importantly, user files should be saved before risky repair attempts. A professional desktop recovery mindset is not built on magic commands. It is built on **stage awareness, backup-first thinking, and calm, tool-based repair**.

Key Takeaways

- **The Ubuntu boot process is easiest to troubleshoot when divided into pre-GRUB, GRUB, and post-GRUB stages.**
- **GRUB is the central boot handoff point** and can be inspected or temporarily edited during startup.
- **Common boot failures often become easier once the exact stage of failure is identified.**
- **`initramfs` is part of the bridge between GRUB and the full Ubuntu system.**
- **Boot-Repair is often the simplest first fix for GRUB-related boot issues.**
- **Live USB rescue workflows are essential** for checking disks, opening files, and repairing systems that no longer boot normally.
- **Always save important user files before deeper repair attempts.**

Part X — Open Source Participation and Ubuntu Ecosystem

Chapter 21 — Ubuntu Community and Open Source Engagement

Ubuntu Desktop is not only a technical platform. It is also part of a much larger **open-source ecosystem** shaped by developers, maintainers, translators, documenters, testers, educators, support volunteers, and everyday users. In the Ubuntu Academy outline, this chapter is explicitly about **community structure, licenses, asking for help, reporting bugs, contributing, and developing good user habits**. That is a very important ending chapter for a desktop book, because it reminds readers that Ubuntu is not just something to consume. It is something people can understand, join, improve, and help sustain.

The attached references support this strongly. *Mastering Ubuntu* emphasizes that one of Ubuntu's strengths is the **community surrounding it**, noting that Ubuntu has many avenues for help, strong documentation, and a wide support culture. It also presents Ubuntu as a free and open-source operating system whose source code can be studied and improved by users. *Ubuntu Linux Bible* adds broader open-source context, including how Linux is built by a worldwide community, how Canonical supports Ubuntu professionally, and how major open-source licenses shape reuse and contribution. *Ubuntu: The Complete Guide* contributes a practical user angle by explaining how Linux communities expect users to search existing documentation and reports first, then ask clearer, better questions.

A useful illustration is to think of Ubuntu as a university campus. The operating system is the campus itself, but the real life of the place comes from the people in it: teachers, librarians, students, technicians, and volunteers. Community engagement means learning how that campus works, where to ask for help, how to respect its rules, and how to contribute something back.

Ubuntu Governance and Community Structure

Ubuntu may feel simple from the desktop side, but behind it stands a structured ecosystem. The Ubuntu Academy outline specifically includes **Ubuntu governance and community structure**, which is a strong reminder that Ubuntu is not just an isolated software package. It is part of a larger coordinated project.

Mastering Ubuntu states that Ubuntu is a free Linux operating system created by the **Canonical community** and repeatedly emphasizes the importance of the community around Ubuntu. It also notes Ubuntu's goals, philosophy, and code-of-conduct-oriented culture in its introductory materials. Meanwhile, *Ubuntu Linux Bible* explains that **Canonical** is the company behind Ubuntu and that it supports Ubuntu through enterprise services, training, and broader professional

solutions built on open-source stacks. This combination is important: Ubuntu has both a **community identity** and an **organizational structure** that helps sustain it.

A practical desktop-level understanding is:

- Ubuntu has a **community side** built on participation, discussion, support, and collaboration.
- Ubuntu also has a **professional support side** through Canonical.
- Users benefit from both: open collaboration and stable stewardship.

A good illustration is to think of Ubuntu governance like a large school system. There are community clubs, student helpers, documentation teams, and volunteer mentors—but there is also administration, official support, and structure to keep everything coherent.

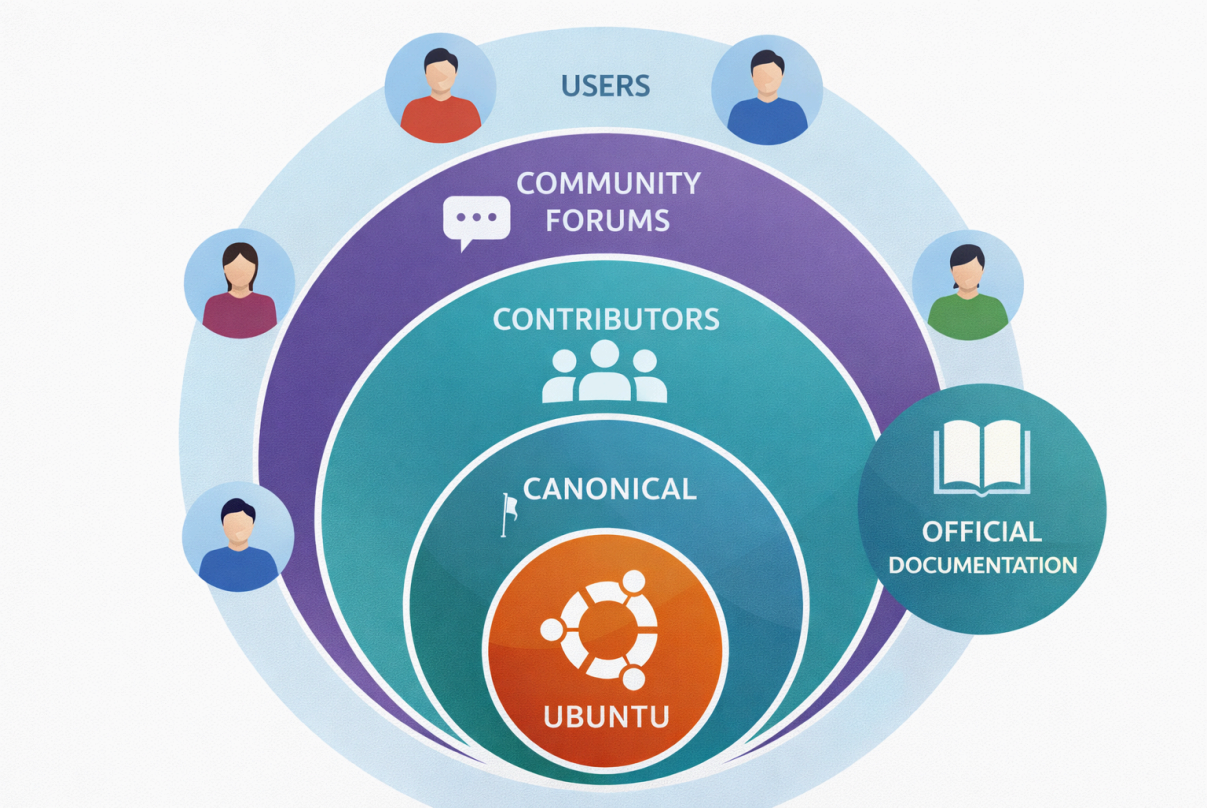


Figure: Concept diagram showing Ubuntu ecosystem layers

Free Software and Open Source Principles

One of Ubuntu's defining characteristics is that it is built on **free software** and **open-source** ideas. *Mastering Ubuntu* directly states that Ubuntu is **free and open-source** and that users can inspect the source code and contribute improvements, learning from the inner workings of the operating system. It also explains that Ubuntu's philosophical goals include **free software**, **open source**, and a broader vision around accessible computing.

The Ubuntu Linux Bible expands this into a broader Linux context. It explains that Linux itself emerged through a worldwide community of developers, that the **Free Software Foundation** helped build many essential Unix-like components, and that the Linux ecosystem is protected by licenses under the broader open-source umbrella. It also emphasizes that the result of open source is software with more freedom to grow and fewer restrictions on how it may be used.

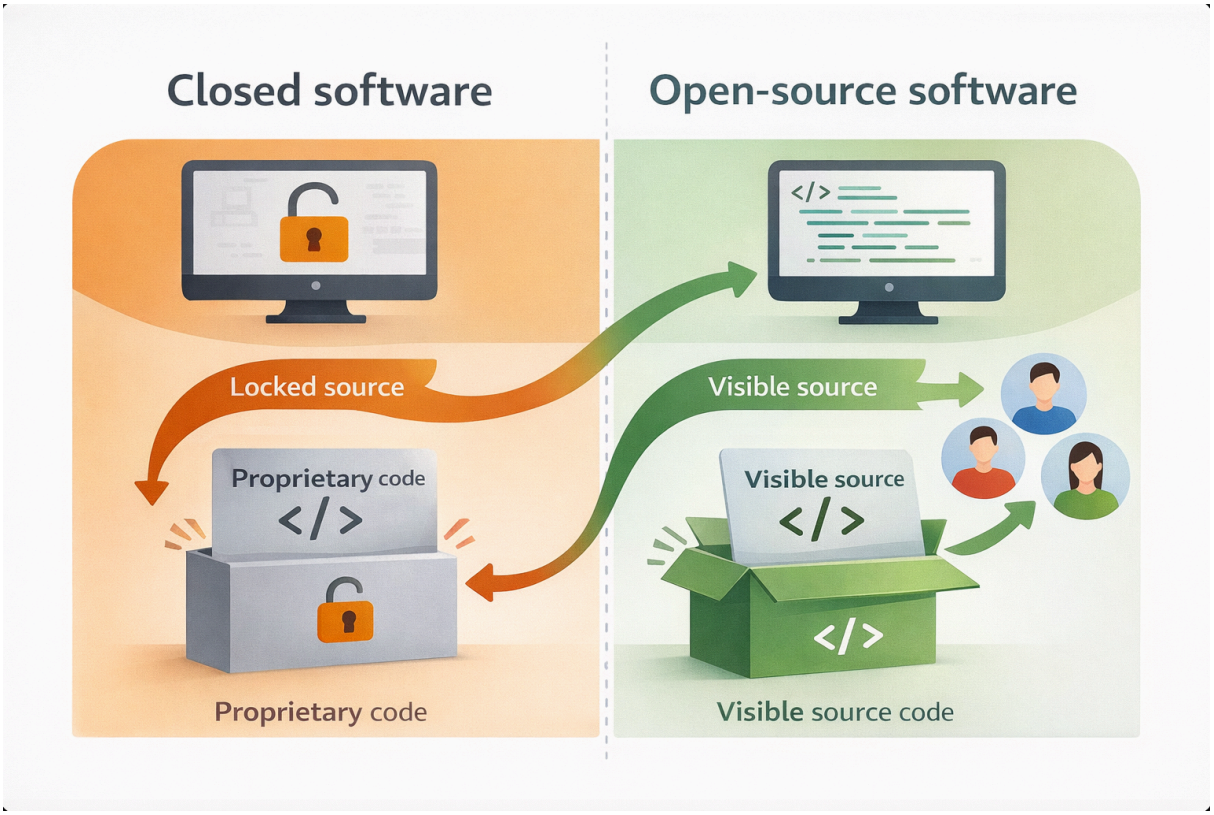
A practical way to explain the difference for desktop readers is:

- **free software** emphasizes user freedom: the freedom to use, study, modify, and share.
- **open source** emphasizes the practical benefits of visible source code, collaboration, and reuse.
- in Ubuntu, these ideas often overlap in real daily use.

A useful illustration is to think of proprietary software as a sealed appliance, while open-source software is more like a workshop design whose plans are visible to the whole community. Users can still just use it, but others can also inspect, improve, and adapt it.

Screenshot notes:

```
[Screenshot-CH21-03: Simple infographic comparing “Closed software” and “Open-source software” with visible source and collaboration arrows.]
[Screenshot-CH21-04: Terminal and source-code editor open together to symbolize inspectable software.]
```


Closed software
Open-source software
Locked source
Visible source
Proprietary code
Visible source
Proprietary code
Visible source code

Common Licenses: GPL, MIT, Apache

The Ubuntu Academy outline explicitly includes **GPL, MIT, and Apache**, asking readers to understand them in terms of **usage, modification, and distribution rights**.

The Ubuntu Linux Bible gives especially useful support for this section. It explains that the **GPL** is the most prominent open-source license in the Linux world and emphasizes that some licenses require changes to remain available to the community when redistributed. It also explains that the **MIT license** is more permissive, describing it as similar to BSD but with fewer extra conditions. In the same section, the book presents open-source licensing as a spectrum of reuse conditions rather than one single rule for all projects.

Because the attached excerpts do not provide a detailed Apache section in the visible snippets, the safest grounded explanation is comparative and conceptual:

- **GPL** is strongly associated with sharing improvements under the same licensing spirit when redistributed.
- **MIT** is a lightweight permissive license that allows broad reuse with fewer conditions.
- **Apache** is also generally discussed in open-source contexts as a permissive license, often favored in modern software ecosystems for practical reuse and contribution workflows.

A good beginner-friendly explanation is this:

- **GPL** = “You may use, study, and modify this, but redistributed modified versions must preserve community-sharing obligations.”
- **MIT** = “You may reuse this very broadly with minimal friction.”
- **Apache** = “You may reuse this broadly too, with a more explicit structured legal framework than very short permissive licenses.”

A useful illustration is to compare software licenses to library-use policies:

- one library lets you borrow and copy notes, but asks you to return improvements to the shared archive,
- another lets you borrow and adapt with very few restrictions,
- another gives broad access under a more detailed formal agreement.

CLI example: inspect license files commonly installed on Ubuntu

```
ls /usr/share/common-licenses
```

Example output:

```
Apache-2.0  Artistic  BSD  GFDL  GPL  GPL-2  GPL-3  LGPL  MIT
```

This is a very useful desktop-administration teaching moment because it shows that license texts are often already installed on Ubuntu systems.

CLI example: read the beginning of the GPL text

```
head -20 /usr/share/common-licenses/GPL-3
```

Example output:

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007
 Copyright (C) 2007 Free Software Foundation, Inc.
```

These commands help readers connect abstract licensing discussion to real files available on their Ubuntu system.

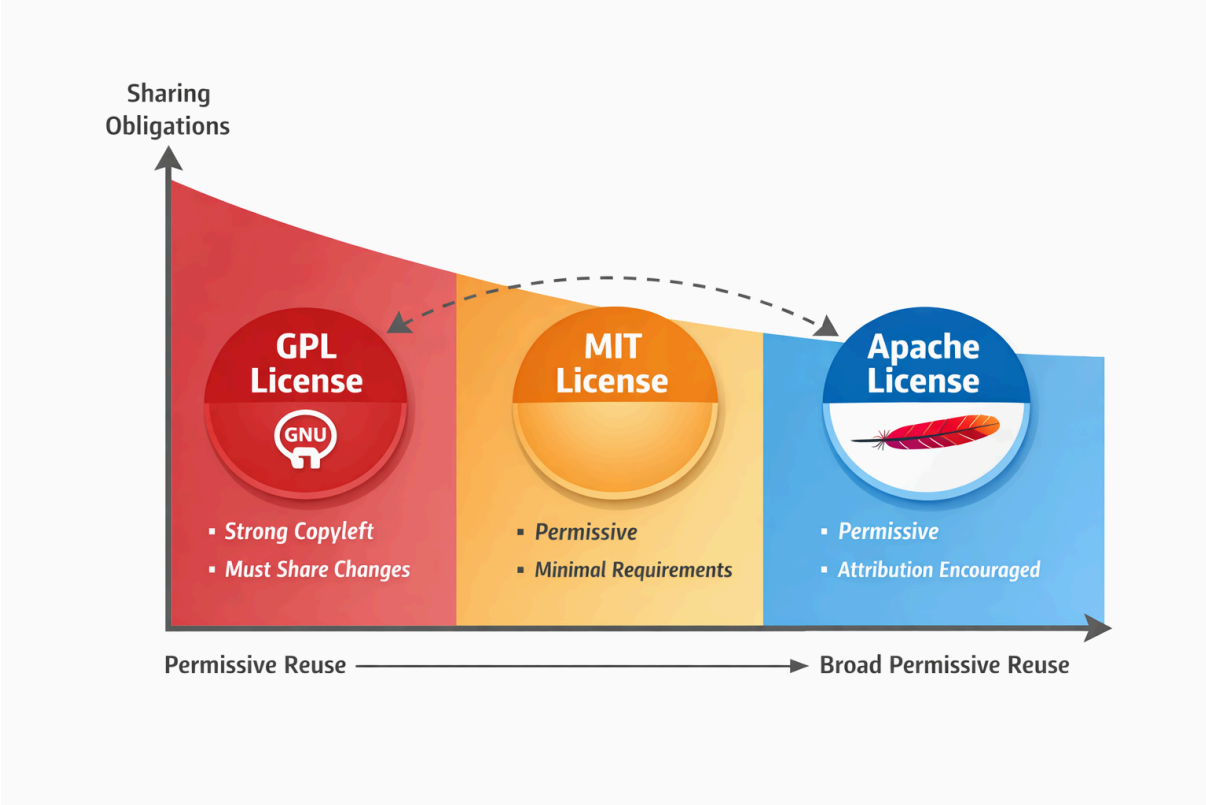


Figure. Diagram comparing GPL, MIT, and Apache at a high level: sharing obligations vs broad permissive reuse

Finding Help

One of Ubuntu's biggest strengths is that users are not alone. *Mastering Ubuntu* explicitly says that Ubuntu has some of the **best support avenues** among Linux distributions because of how widely it is discussed and how many people regularly talk about it. It also emphasizes Ubuntu's strong documentation culture. *Ubuntu: The Complete Guide* adds an important cultural lesson: when users need help, they should first check whether the problem has already been answered, reported, or documented. Communities respond better when users show that they have already tried to learn responsibly.

A healthy Ubuntu help-seeking habit is:

- look for the answer in the **official documentation** first
- search existing questions and bug reports
- ask clearly and specifically when documentation is not enough
- include the actual error, version, and relevant context
- be respectful of volunteer time

A good illustration is to think of help-seeking like visiting a library reference desk. The librarian is happy to help, but it works best when you arrive with a clear question, not just "something is wrong."

Official documentation

The Ubuntu Academy outline explicitly names **help.ubuntu.com** and official Ubuntu Desktop documentation as first-class resources. *Ubuntu: The Complete Guide* supports this strongly by saying Ubuntu provides basic built-in help and that many applications provide their own built-in help as well. It also notes that pressing **F1** often opens application help and that Help → Contents is a standard entry point.

CLI example: use built-in command help

```
ls --help | less
```

Example output:

```
Usage: ls [OPTION]... [FILE]...
List information about the FILEs...
```

Ubuntu: The Complete Guide explicitly recommends the `--help` pattern and piping output into `less` when it is too long.

CLI example: use a man page

```
man ls
```

Example output:

```
LS(1)              User Commands               LS(1)
NAME
       ls - list directory contents
```

This fits naturally with the Ubuntu Academy resource outline, which includes **man pages** as part of finding help.

Ask Ubuntu

The Ubuntu Academy outline explicitly names **Ask Ubuntu** as a recommended place to ask questions. This is especially valuable because it provides searchable Q&A in a structured format. It is ideal when a user wants a focused answer to a concrete desktop or package-management problem.

Ubuntu Discourse

The Ubuntu Academy outline also explicitly includes **Ubuntu Discourse**, which is useful for broader discussion, announcements, and community-guided help rather than only narrow troubleshooting questions.

Forums and chat communities

The same outline also mentions **forums** and **IRC/Discord-style chat communities**, which reflects Ubuntu's broader support culture. *Ubuntu: The Complete Guide* reinforces that Linux communities often form around forums, standard documentation, social channels, and community discussion spaces.



Figure. Mockup diagram of help sources

Reporting Bugs and Giving Useful Feedback

Good communities do not only answer questions. They also improve software. The Ubuntu Academy outline explicitly includes **reporting bugs** and names **Launchpad** as the contribution entry point for this kind of work.

Ubuntu: The Complete Guide provides a crucial cultural reminder for bug reporting: before asking for help or demanding fixes, users should check whether the issue is already documented or already reported. This is not bureaucracy. It is respect for maintainers and for the quality of the support process.

Useful bug reports usually include:

- what the user expected
- what actually happened
- version information
- exact steps to reproduce
- screenshots or logs when relevant
- whether the issue is consistent or intermittent

A good illustration is to think of bug reporting like writing a lab incident report. “It broke” is not enough. The report needs enough detail that someone else can repeat the situation and see the same failure.

CLI example: gather Ubuntu version info for a bug report

```
lsb_release -a
```

Example output:

```
Distributor ID: Ubuntu
Description:    Ubuntu 24.04 LTS
Release:        24.04
Codename:       noble
```

CLI example: gather kernel version info

```
uname -r
```

Example output:

```
6.8.0-52-generic
```

CLI example: capture recent session logs

```
journalctl -b -n 20
```

Example output:

```
Mar 14 09:50:01 ubuntu-desktop gnome-shell[2412]: Window manager
warning: ...
Mar 14 09:50:12 ubuntu-desktop systemd[1]: Started User Manager
for UID 1000.
```

These commands are exactly the kind of groundwork that makes feedback more useful and more respectful.

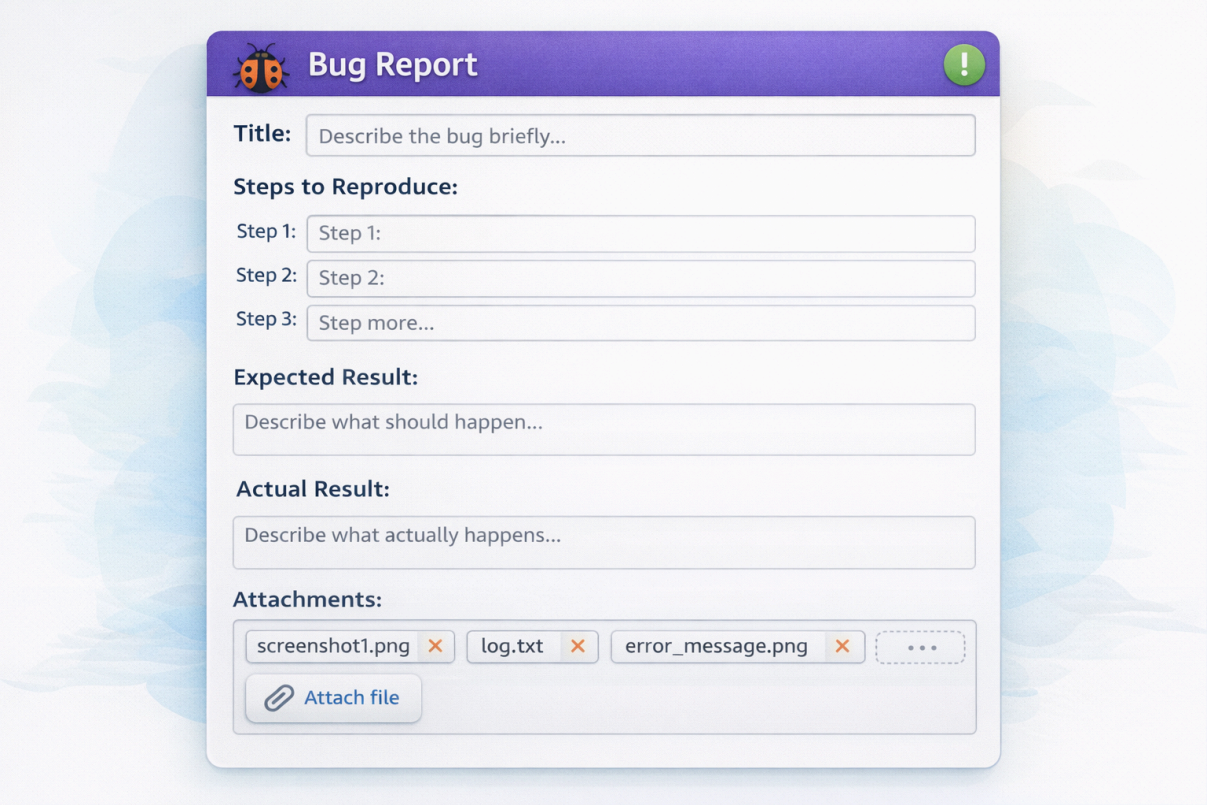


Figure. Simple Launchpad-style bug-report

Contributing Through Testing, Documentation, Translation, and Advocacy

One of the most encouraging truths about Ubuntu is that contribution is **not limited to writing code**. The Ubuntu Academy outline explicitly names **testing, documentation, translation, and advocacy** as valid ways to participate. *Mastering Ubuntu* also emphasizes that Ubuntu users can contribute because the system is open source and because community participation is part of Ubuntu's identity.

This is important because many desktop users assume:
"I am not a programmer, so I cannot contribute."

That is false.

A practical contribution map looks like this:

- **testing** → try a feature, reproduce bugs, report what happens
- **documentation** → improve instructions, clarity, examples, and screenshots
- **translation** → help make Ubuntu and its documentation accessible in more languages
- **advocacy** → teach others, introduce Ubuntu responsibly, share best practices, support new users

Mastering Ubuntu also notes that open-source communities improve because many users, companies, and contributors return fixes and features to the ecosystem. It presents community support and openness as part of why Ubuntu remains strong.

A useful illustration is to think of Ubuntu contributions like caring for a shared garden. Some people write code like planting trees. Others write documentation like placing labels on paths. Others translate like adding signs in more languages. Others welcome newcomers so they know where to begin.

CLI example: inspect current locale before thinking about translation-related context

```
locale
```

Example output:

```
LANG=en_US.UTF-8
LANGUAGE=en_US:
LC_TIME=en_US.UTF-8
LC_ALL=
```

CLI example: gather system information when testing a bug or feature

```
hostnamectl
```

Example output:

```
  Static hostname: ubuntu-desktop
 Operating System: Ubuntu 24.04 LTS
           Kernel: Linux 6.8.0-52-generic
```

These simple commands help readers understand that even non-coding contributions often benefit from careful technical context.

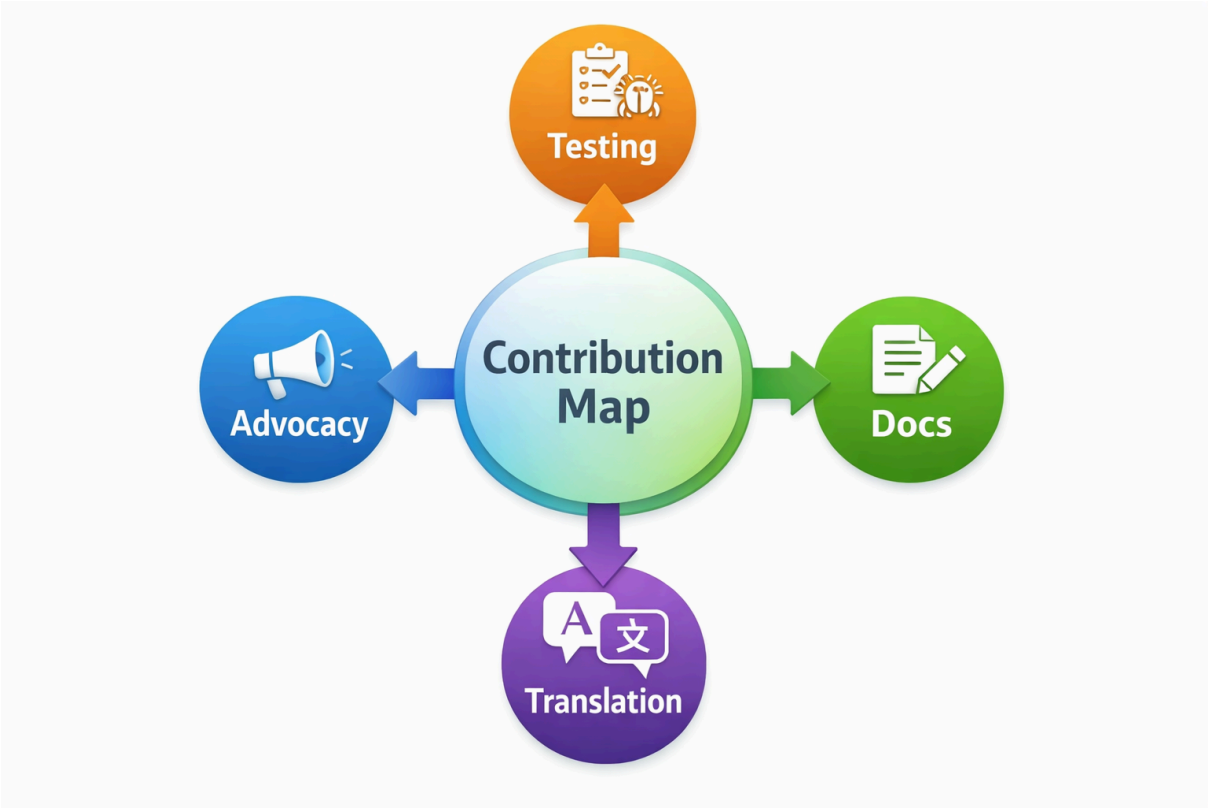


Figure. Contribution map diagram

Building Good Habits as an Ubuntu Desktop User

The final goal of this chapter is not only to introduce community resources, but to help readers become the kind of Ubuntu users who strengthen the ecosystem rather than drain it. *Ubuntu: The Complete Guide* is especially valuable here because it explains the social expectations of open-source help communities: users should search first, read the documentation, and ask clearly. *Mastering Ubuntu* complements this by highlighting Ubuntu's culture of help, support, documentation, security updates, and open participation.

Good Ubuntu Desktop habits include:

- search before asking
- read built-in help and official docs first
- prefer trusted repositories and safe update habits
- write clear notes when something breaks
- save useful commands and solutions for future reference
- help others when you are able
- contribute back when you benefit from the ecosystem

A useful illustration is to think of a good Ubuntu user as a responsible member of a workshop. They borrow tools carefully, return them in good condition, label what they learn, and help the next person use the space safely.

CLI example: use built-in help for a command

```
nmcli --help | less
```

Example output:

```
Usage: nmcli [OPTIONS] OBJECT { COMMAND | help }
```

CLI example: save troubleshooting notes for yourself

```
mkdir -p ~/Documents/ubuntu-notes
echo "Use 'journalctl -b -n 20' for quick boot/session logs." >>
~/Documents/ubuntu-notes/troubleshooting.txt
cat ~/Documents/ubuntu-notes/troubleshooting.txt
```

Example output:

```
Use 'journalctl -b -n 20' for quick boot/session logs.
```

This small example reinforces a powerful habit: Ubuntu learning compounds when users keep notes and turn one solved issue into future confidence.



Figure. Good Ubuntu User Habits

Guided Community and Contribution Lab

To make this chapter practical, readers can complete a short engagement exercise:

- open the built-in **Help** application from Ubuntu Desktop
- use `man ls` and `ls --help | less`
- identify the Ubuntu version with `lsb_release -a`
- create a small personal **Ubuntu notes** folder
- write a sample bug-report outline with:
  - title
  - steps to reproduce
  - expected result
  - actual result
- compare the roles of **Ask Ubuntu**, **Ubuntu Discourse**, and official docs
- list one possible contribution in each category:
  - testing
  - documentation
  - translation
  - advocacy

This kind of exercise helps readers see that community engagement is not abstract. It is a practical extension of using Ubuntu well.

Chapter Reflection

Ubuntu Community and Open Source Engagement reminds readers that Ubuntu is more than a desktop product. It is part of a living open-source ecosystem. Ubuntu's community structure combines volunteer participation, strong documentation, and broader organizational support from Canonical. Free software and open-source principles give users freedom to study, modify, and share, while licenses such as GPL, MIT, and Apache define how that freedom operates in practice. Good Ubuntu users know where to find help: official documentation, built-in help, Ask Ubuntu, Ubuntu Discourse, forums, and chat communities. They also understand how to report bugs responsibly and how to contribute in many ways beyond code—through testing, writing, translation, and advocacy. Most importantly, they build habits that respect the time, knowledge, and generosity of the people who keep the ecosystem alive.

Key Takeaways

- **Ubuntu is both a technical platform and a community ecosystem.**
- **Canonical provides structure and support, while the wider community provides collaboration and knowledge sharing.**
- **Free software and open source** are foundational to Ubuntu's identity.
- **GPL, MIT, and Apache** represent different licensing approaches to reuse and redistribution.
- **Good help-seeking starts with official docs and existing answers before asking the community.**
- **Ask Ubuntu, Ubuntu Discourse, forums, and chat communities** serve different but complementary support roles.
- **Contribution is not limited to coding**; testing, documentation, translation, and advocacy all matter.
- **A good Ubuntu Desktop user develops habits that strengthen the ecosystem for others.**

Part XI — Productivity, Practice, and Real-World Use

Chapter 22 — Everyday Desktop Best Practices

A good Ubuntu Desktop user does not rely only on features. They rely on **habits**. The Ubuntu Academy outline places this chapter at the end for a good reason: after learning installation, applications, networking, security, maintenance, troubleshooting, and community engagement, the real question becomes, **how should a person use Ubuntu well every day?** The outline explicitly highlights **workflow optimization with shortcuts and workspaces, file discipline, safe update habits, low-end PC awareness, and stable yet flexible desktop practice**.

The attached references support this strongly. *Mastering Ubuntu* emphasizes that Ubuntu is customizable, secure, backed by a large open-source community, and suitable for everyday browsing, email, office work, and multimedia. *Ubuntu: The Complete Guide* gives practical examples of **workspaces**, launcher behavior, and desktop customization that improve daily use. *Ubuntu Linux Bible* reinforces the GNOME workflow model, showing that Ubuntu's default desktop is designed around **Activities**, **search**, **workspaces**, and practical keyboard-driven use. Together, these sources support a chapter that is less about one command or one app and more about developing a sustainable way to work.

A useful illustration is to think of Ubuntu Desktop like a study room. A beginner may only see the chair and the table. A skilled user arranges the room so that the light is right, the books are in order, the most-used tools are within reach, and interruptions are minimized. Everyday best practices are about turning Ubuntu from "a system that works" into **a system that helps you work well**.

Productive Workflows with Workspaces and Shortcuts

One of GNOME's biggest strengths is that it encourages a **focused workflow** instead of a cluttered one. The attached references repeatedly support this idea. *Mastering Ubuntu* describes the **Activities Overview**, the **Dash**, **search**, and the **workspace list** as key parts of GNOME. *Ubuntu: The Complete Guide* gives a clear explanation of **virtual desktops/workspaces**, noting that they help keep the desktop organized by grouping related applications together. *Ubuntu Linux Bible* also includes hands-on exercises that move Nautilus and Firefox between workspaces and use the keyboard to switch efficiently.

A practical desktop workflow benefits from:

- using **workspaces** to separate different tasks
- learning a small set of **keyboard shortcuts**
- using **search** instead of hunting through menus
- keeping only essential applications pinned to the Dock
- avoiding too many open windows on one screen at once

A simple and powerful workspace strategy is:

- **Workspace 1** → communication and browser
- **Workspace 2** → writing, office work, coding, or reading
- **Workspace 3** → file management, downloads, and media
- **Workspace 4** → optional support, reference, or secondary task

A good illustration is to think of workspaces as separate desks in the same room. One desk is for research, one for writing, one for files, and one for temporary tasks. The room stays calmer because every object has a place.

CLI example: confirm the current desktop environment

```
echo $XDG_CURRENT_DESKTOP
echo $DESKTOP_SESSION
```

Example output:

```
ubuntu:GNOME
ubuntu
```

CLI example: launch Terminal quickly for a desktop workflow

```
gnome-terminal &
```

Example output:

```
[1] 8015
```

CLI example: inspect currently running graphical apps

```
ps -ef | grep -E 'firefox|nautilus|gnome-text-editor' | grep -v grep
```

Example output:

```
student    4121  2980  0 10:22 ?        00:00:01 nautilus
student    4178  2980  0 10:22 ?        00:00:00 gnome-text-editor
student    4230  2980  2 10:22 ?        00:00:03 firefox
```

This reminds readers that desktop workflows are still backed by real processes, and that organized work is not only visual but also systematic.

Useful shortcuts readers should actually practice every day:

- **Super** → open Activities
- **Ctrl + Alt + T** → open Terminal
- **Alt + Tab** → switch applications
- **Super + Left / Right** → tile windows
- **Super + PgUp / PgDn** → switch workspaces
- **Super + Shift + PgUp / PgDn** → move windows between workspaces

Screenshot notes:

[Screenshot-CH22-01: Activities Overview showing multiple workspaces used intentionally for different tasks.]
[Screenshot-CH22-02: Ubuntu desktop with browser on one workspace and Files on another, demonstrating reduced clutter.]
[Screenshot-CH22-03: Split-screen layout using tiling shortcuts for browser and writer/editor side by side.]

File Discipline and Backup Habits

A productive Ubuntu Desktop is not built only through good software. It is also built through **good file habits**. The Ubuntu Academy outline repeatedly highlights file search, practical file management, backup basics, and recovery planning. That means everyday best practice must include **how users name, store, clean, and protect their files**.

A disciplined desktop user should avoid treating **Downloads**, **Desktop**, and **Documents** as one giant pile. A practical file habit system includes:

- move downloaded files out of **Downloads** quickly
- keep the **Desktop** visually clean
- create clear project folders with meaningful names
- archive finished work instead of mixing it with active work
- back up important folders regularly
- test restore, not just backup creation

A useful illustration is to think of your files like papers on a real office desk. If everything stays in one stack forever, even simple work becomes stressful.

CLI example: create a structured workspace folder tree

```
mkdir -p ~/Workspace/{Active,Archive,Share,Screenshots,Backups}
tree -d ~/Workspace
Example output:
/home/student/Workspace
├── Active
├── Archive
├── Backups
├── Screenshots
└── Share
```

CLI example: move project work into an active workspace

```
mv ~/Documents/Ubuntu-Academy ~/Workspace/Active/
ls ~/Workspace/Active
Example output:
Ubuntu-Academy
```

CLI example: prepare a safe backup location

```
mkdir -p ~/Workspace/Backups/DesktopBackup
ls ~/Workspace/Backups
Example output:
DesktopBackup
```

This reflects the broader backup discipline already supported by the Déjà Dup and maintenance chapters. A desktop user who works productively today should also make sure today's work still exists tomorrow.

Screenshot notes:

[Screenshot-CH22-04: Files window showing a clean workspace structure: Active, Archive, Share, Screenshots, Backups.]
[Screenshot-CH22-05: Déjà Dup or backup tool configured to back up Documents, Pictures, and project folders.]

Safe Update Habits

Ubuntu's update system is one of its strengths, but good users still need good habits. The Ubuntu Academy outline explicitly emphasizes **updates for security patching**, **manual and automatic update workflows**, and **unattended-upgrades**. *Ubuntu: The Complete Guide* explains that urgent fixes can happen quietly in the background while broader updates are offered at a sensible time. *Ubuntu Linux Bible* reinforces that prompt patching is an important security practice.

A practical safe update habit is:

- apply **security updates promptly**
- do normal package updates regularly
- avoid doing major updates in the middle of urgent work
- back up or snapshot before larger changes
- read warnings if graphics, kernels, boot components, or storage layers are involved

A useful analogy is to think of updates like building maintenance. Replacing a broken lock should happen quickly. Renovating half the building should happen at the right time, with preparation.

CLI example: refresh package metadata

```
sudo apt update
```

Example output:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
```

CLI example: install upgrades safely

```
sudo apt upgrade -y
```

Example output:

```
Reading package lists... Done
Building dependency tree... Done
Calculating upgrade... Done
The following packages will be upgraded:
  firefox linux-firmware
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

CLI example: inspect upgradeable packages first

```
apt list --upgradable
```

Example output:

```
Listing... Done
firefox/noble-updates 1:1snap1-0ubuntu2 amd64 [upgradable from:
1:1snap1-0ubuntu1]
```

A stable Ubuntu Desktop is not a desktop that never changes. It is a desktop where changes happen carefully and intentionally.

Screenshot notes:

[Screenshot-CH22-06: Software Updater showing available updates and a user choosing an appropriate time to apply them.]
[Screenshot-CH22-07: Terminal showing apt update, apt list --upgradable, and apt upgrade in sequence.]

Resource-Aware Desktop Usage on Low-End PCs

Not every Ubuntu Desktop runs on a powerful workstation. The attached references acknowledge this reality clearly. *Mastering Ubuntu* notes that different desktop environments vary in performance and that some systems with modest resources may struggle with heavier configurations. The same source also remarks that KDE, while powerful and flexible, is not ideal for low-end CPUs in some cases. *Ubuntu Linux Bible* reminds readers that Ubuntu can also use lighter desktops such as **LXDE** or **Xfce** when appropriate.

This is an important best-practice lesson: a good Ubuntu user should match the desktop to the hardware rather than insisting that every machine behave like a modern high-end laptop.

A resource-aware Ubuntu habit set includes:

- keep the number of startup applications low
- avoid excessive GNOME extensions
- reduce unnecessary animation or visual extras
- keep browser tabs under control
- choose lightweight apps when appropriate
- consider a lighter desktop environment if the hardware is truly constrained

A useful analogy is to think of low-end desktop usage like packing for a small backpack. The goal is not to carry everything possible. It is to carry what is essential and useful.

CLI example: inspect memory usage

```
free -h
```

Example output:

```
               total        used        free      shared  buff/cache   available
Mem:           3.8Gi       1.7Gi       1.1Gi       210Mi       1.0Gi       1.8Gi
Swap:          2.0Gi          0B       2.0Gi
```

CLI example: inspect disk and workload

```
df -h
uptime
```

Example output:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       118G   29G   84G  26% /
 10:44:08 up 2:12,  1 user,  load average: 0.41, 0.50, 0.46
```

CLI example: inspect startup entries

```
ls ~/.config/autostart
```

Example output:

```
update-notifier.desktop
```

This simple output may actually be a sign of good discipline on a lower-end machine: fewer background startup items usually means less startup pressure.

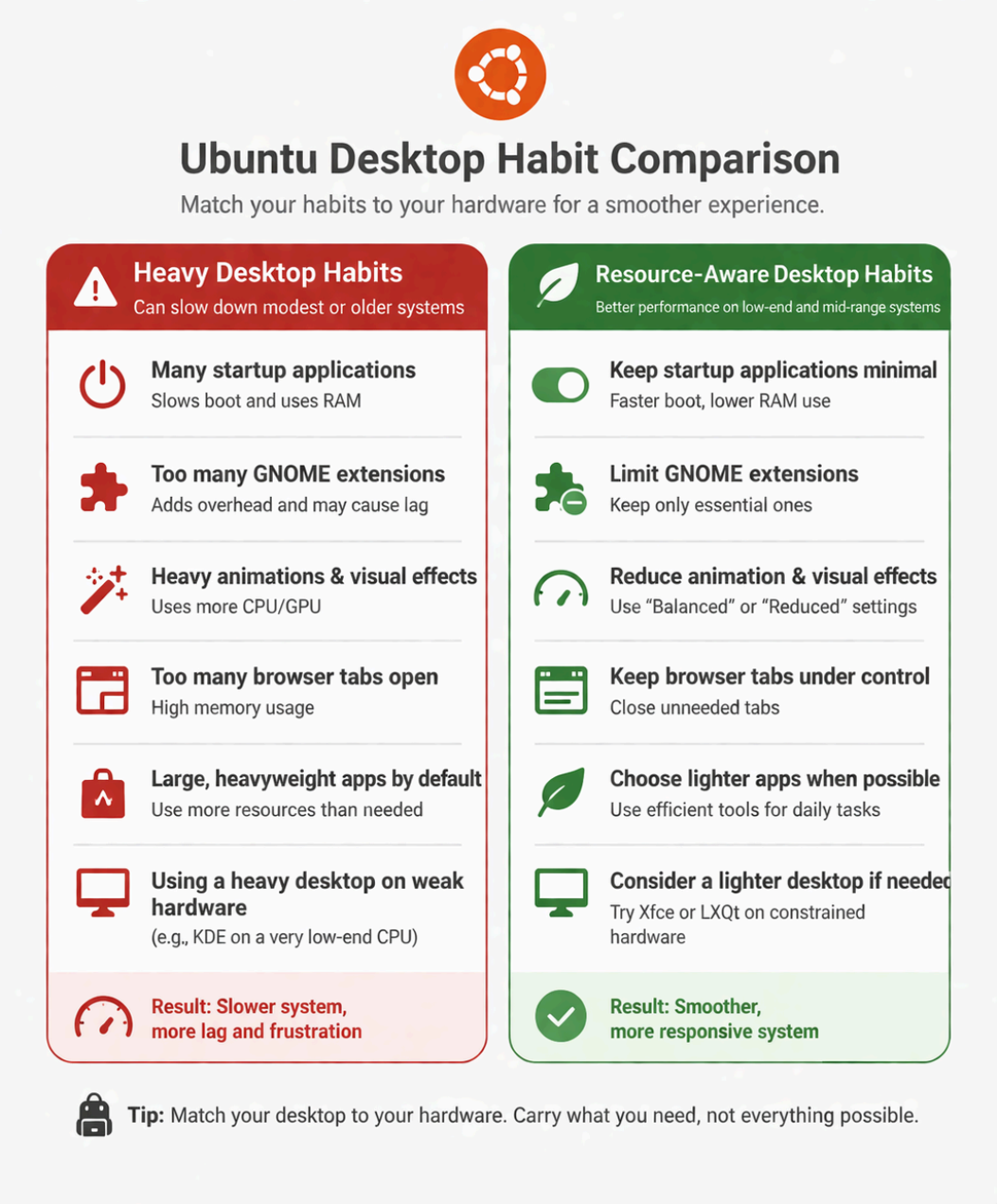


Figure. “Heavy desktop habits” vs “Resource-aware desktop habits”

Best Practices for Students, Teachers, Office Users, and Home Users

Ubuntu Desktop is flexible because different people use it differently. The Ubuntu Academy outline explicitly mentions practical desktop operations and best practices across real-world use, and the attached references show Ubuntu as a platform for **browsing, email, multimedia, office work, education, and customization**. They also explicitly mention **Edubuntu** as an education-oriented flavor and describe Ubuntu Desktop as suitable for daily user activities such as web browsing, email, and media.

That makes it useful to describe best practices by user type.

Students

Students benefit most from:

- clear project folders for each course
- regular backups of assignments
- using workspaces for reading vs writing
- keeping screenshots and PDF notes organized
- avoiding last-minute updates right before deadlines

A good student habit is to treat the laptop like a study notebook: organized, backed up, and ready before the exam or submission.

Teachers

Teachers benefit from:

- a clean presentation workspace
- stable browser and office tools
- external-display readiness
- a dedicated folder system for lecture slides, recordings, and student materials
- backups before modifying teaching content

A useful teaching analogy is to think of Ubuntu as a digital classroom desk: it should be calm, clear, and predictable when class begins.

Office users

Office users benefit from:

- consistent update windows outside working peaks
- careful file naming and document versions
- reliable printers and network shares
- restrained customization for stability
- strong passwords and locked sessions on shared desks

A good office habit is to prioritize reliability over novelty.

Home users

Home users benefit from:

- simple backup routines
- safe browsing and update habits
- media and communication tools kept current
- limited admin changes unless necessary
- a balance between convenience and security

A good home-user analogy is to think of Ubuntu as a family appliance: it should remain easy, safe, and understandable for daily life.

CLI example: create role-based folders

```
mkdir -p ~/Roles/{Student,Teacher,Office,Home}
ls ~/Roles
```

Example output:

```
Home  Office  Student  Teacher
```

This small example helps readers think structurally about how roles shape workflow and organization.

Maintaining a Stable Yet Flexible Ubuntu Desktop

One of Ubuntu's greatest strengths is that it can be both **stable** and **customizable**. The challenge is balance. The attached references support both sides of this. *Mastering Ubuntu* praises Ubuntu's customizability and GNOME's flexibility. *Ubuntu: The Complete Guide* explains how users can customize the launcher, workspaces, and desktop appearance. But the broader Ubuntu Academy outline also repeatedly emphasizes **safe software sources, performance tuning, extension awareness, updates, and troubleshooting**—all reminders that flexibility should not become chaos.

A stable-yet-flexible Ubuntu Desktop usually follows these principles:

- customize intentionally, not impulsively
- prefer official repositories first
- add GNOME extensions only when they solve a real problem
- keep the Dock useful, not crowded
- document unusual changes
- update regularly, but thoughtfully
- back up before larger changes
- remove what you no longer use

A useful illustration is to think of Ubuntu customization like tuning a musical instrument. A little tuning makes it better. Constant unnecessary adjustment can make it unstable.

CLI example: inspect enabled GNOME extensions

```
gnome-extensions list --enabled
```

Example output:

```
ubuntu-dock@ubuntu.com
ding@rastersoft.com
```

This is a useful quick check because too many extensions can sometimes become a stability or performance problem.

CLI example: inspect user startup items

```
ls ~/.config/autostart
```

Example output:

```
update-notifier.desktop
```

CLI example: keep notes on non-default changes

```
mkdir -p ~/Documents/ubuntu-notes
echo "Enabled dark mode, pinned Firefox/Files/LibreOffice, added
one workspace helper extension." >>
~/Documents/ubuntu-notes/desktop-changes.txt
cat ~/Documents/ubuntu-notes/desktop-changes.txt
```

Example output:

```
Enabled dark mode, pinned Firefox/Files/LibreOffice, added one
workspace helper extension.
```

This kind of note-taking habit is simple, but it is one of the best ways to keep a customized system understandable months later.



Figure. comparison of a cluttered unstable desktop and a stable well-organized desktop

Guided Everyday Best Practices Lab

To make this chapter practical, readers can complete a short routine-building exercise:

- create a structured **Workspace** folder tree
- move one real project into an **Active** folder
- assign at least two **workspaces** for different task types
- practice **Super**, **Alt+Tab**, and tiling shortcuts
- run `apt list --upgradable` and schedule a safe update moment
- inspect `~/.config/autostart` and remove unnecessary startup items
- check memory and disk usage with `free -h` and `df -h`
- write one short note in a personal **ubuntu-notes** folder describing recent system changes
- review whether current customizations are helping productivity or only adding noise

This kind of exercise turns the chapter into daily discipline rather than just advice.

Chapter Reflection

Everyday Desktop Best Practices brings the entire Ubuntu Academy workflow together. Productive Ubuntu use depends on intentional habits: organizing work with workspaces and shortcuts, maintaining disciplined file structures, backing up important data, applying updates safely, and adjusting usage to match the available hardware. Different user groups—students, teachers, office users, and home users—need slightly different workflow priorities, but all benefit from the same core principles: clarity, consistency, restraint, and preparation. The strongest Ubuntu Desktop is not the most customized or the most technically impressive. It is the one that stays **stable enough to trust, flexible enough to adapt, and organized enough to support real daily work**.

Key Takeaways

- **Good Ubuntu use is built on habits, not only on features.**
- **Workspaces and shortcuts** reduce clutter and improve focus.
- **File discipline and backup routines** are part of productivity, not separate from it.
- **Safe updates** are regular, intentional, and supported by backup awareness.
- **Low-end PCs benefit from lighter habits**, fewer background items, and restrained customization.
- **Students, teachers, office users, and home users** all benefit from Ubuntu, but their workflow priorities differ.
- **A stable yet flexible Ubuntu Desktop** comes from intentional customization, not random tweaking.

Chapter 23 — Guided Hands-On Labs

This chapter transforms the Ubuntu Academy journey from reading into **doing**. The uploaded Ubuntu Academy outline explicitly states that the companion book should include **hands-on labs and exam-style exercises**, including scenario-based tasks such as **installing Flatpak apps, configuring Wi-Fi with firewall rules, fixing GRUB, and customizing GNOME**. It also emphasizes that the real Canonical Academy module is **interactive, self-paced, and performance-based**, which makes a structured lab chapter especially appropriate.

The supporting reference books reinforce this approach. *Ubuntu: The Complete Guide* includes practical installation, VirtualBox, boot repair, and rescue-media workflows. *Mastering Ubuntu* provides desktop-oriented installation, GNOME usage, package installation, Flatpak, Snap, and `.deb` examples. *Ubuntu Linux Bible* adds stronger administrative grounding around APT, GRUB, CUPS, AppArmor, updates, and recovery. Together, these books support a lab chapter that is **practical, progressive, and realistic for students, teachers, office workers, and aspiring administrators**.

A useful illustration is to think of these labs as a flight simulator for Ubuntu Desktop. Earlier chapters explained the controls. This chapter lets readers **practice real scenarios safely**, one skill stack at a time.

How to Use These Labs

Each lab is designed with the same structure so learners can repeat the habit:

- **Goal** — what the learner is trying to achieve
- **Skills covered** — the main Ubuntu Desktop competencies involved
- **Preparation** — what should already exist before starting
- **Tasks** — what to do, step by step
- **Verification** — how to confirm success
- **Reflection** — what the learner should understand afterward

A good lab is not just about finishing tasks. It is about building the habit of checking results, documenting changes, and learning how Ubuntu behaves in the real world. The Ubuntu Academy outline itself recommends **VMs, dual-boot, and live USB**

testing as lab setup approaches, which strongly supports using these exercises in practice rather than only as reading material.

Lab 1: Install Ubuntu Desktop in a VM

Goal

Install Ubuntu Desktop inside a virtual machine and reach a usable GNOME desktop session.

Skills covered

- VirtualBox setup
- ISO-based installation
- first boot
- post-install updates
- basic desktop navigation

Why this lab matters

Ubuntu: The Complete Guide gives a practical VirtualBox installation workflow: download VirtualBox, create a VM, assign RAM and disk space, attach the Ubuntu ISO, and boot into the live session to install Ubuntu. It even recommends useful details such as enabling 3D acceleration and using a disk size larger than the low default.

Preparation

- Host computer with VirtualBox installed
- Ubuntu Desktop ISO
- At least 20–32 GB available disk space
- At least 2–4 GB RAM available for the VM

Tasks

1. Create a new VM named **Ubuntu Desktop Lab**.
2. Assign memory:
    - minimum: **2048 MB**
    - preferred: **4096 MB** when possible.
3. Create a virtual disk:
    - minimum: **20 GB**
    - better: **32 GB** or more.
4. Attach the Ubuntu ISO.
5. Start the VM and boot the ISO.
6. Choose **Try or Install Ubuntu**.
7. Complete the desktop installation using:
    - full name
    - username

    - password
    - timezone
8. Reboot into the installed system.
9. Run first updates:

```
sudo apt update
sudo apt upgrade -y
```

Verification

```
echo $XDG_CURRENT_DESKTOP
lsb_release -a
```

Example output:

```
ubuntu:GNOME
Distributor ID: Ubuntu
Description:    Ubuntu 24.04 LTS
Release:        24.04
Codename:       noble
```

Reflection

After this lab, the learner should understand that Ubuntu Desktop can be tested safely inside a VM, which is exactly the kind of lab-friendly environment recommended in the Ubuntu Academy outline.

Lab 2: Configure a Full Daily-Use Workstation

Goal

Turn a fresh Ubuntu Desktop installation into a stable everyday workstation for browsing, writing, files, and communication.

Skills covered

- connecting internet
- updating
- installing first applications
- organizing files
- preparing backups

Why this lab matters

The attached references describe Ubuntu Desktop as suitable for **web browsing, email, multimedia, and office work**, and they also emphasize that a newly installed desktop should be brought up to date and equipped with practical applications before regular use.

Preparation

- A working Ubuntu Desktop install
- Internet access

Tasks

1. Connect to Wi-Fi or Ethernet.
2. Update the system.
3. Install a basic workstation set:

   ```
   sudo apt install libreoffice thunderbird vlc gnome-tweaks
   ```

4. Create a useful workspace structure:

   ```
   mkdir -p ~/Workspace/{Active,Archive,Share,Screenshots,Backups}
   tree -d ~/Workspace
   ```

5. Open Firefox, Files, and LibreOffice.
6. Pin the most-used apps to the Dock.
7. Open **Déjà Dup** or prepare a backup folder.
8. Set a wallpaper and choose light or dark mode.

Verification

```
ls ~/Workspace
```

Example output:

```
Active  Archive  Backups  Screenshots  Share
```

Reflection

This lab helps readers experience Ubuntu as a real workstation rather than just a successful installation.

Lab 3: Install Apps via APT, Snap, Flatpak, and `.deb`

Goal

Install software using four different Ubuntu-compatible methods and understand when each one is appropriate.

Skills covered

- APT
- Snap
- Flatpak
- `.deb` via `dpkg`
- software-source awareness

Why this lab matters

The Ubuntu Academy outline explicitly lists:

- `apt update/upgrade/install/remove/search`
- `dpkg`
- Snap basics
- Flatpak with Flathub
- safe handling of third-party packages.

Mastering Ubuntu provides concrete examples of installing apps through **APT**, **Snap**, **Flatpak**, and **DEB** packages.

Tasks

1. Install an APT app:

```
sudo apt install vlc
```

2. Install a Snap app:

```
sudo snap install foliate
```

3. Enable Flatpak and Flathub:

```
sudo apt install flatpak
flatpak remote-add --if-not-exists flathub
https://flathub.org/repo/flathub.flatpakrepo
```

4. Install a Flatpak app:

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

5. Download a trusted `.deb` package and install it:

```
sudo dpkg -i example-package.deb
```

Verification

```
snap list
flatpak list
dpkg -l | head
```

Reflection

This lab teaches that Ubuntu software management is flexible, but source trust still matters. The Ubuntu Linux Bible explicitly warns that PPAs and other outside sources are not curated like mainstream repositories.

Lab 4: Customize GNOME for Productivity

Goal

Adjust GNOME so the desktop becomes cleaner, faster to use, and better suited to focused work.

Skills covered

- Appearance
- dark mode
- GNOME Tweaks
- extensions
- shortcuts
- workspaces

Why this lab matters

The Ubuntu Academy outline directly calls out:

- themes, icons, wallpapers
- GNOME Extensions
- GNOME Tweaks
- keyboard shortcuts
- dock/panel adjustments.

Mastering Ubuntu also includes GNOME interface, shell extensions, shortcuts, and GSettings discussion.

Tasks

1. Install GNOME Tweaks if needed.
2. Change wallpaper.
3. Enable dark mode.
4. Pin only essential applications.
5. Create at least two workflow workspaces.
6. Install or enable one useful extension only.
7. Practice:
    - **Super**
    - **Alt+Tab**
    - **Ctrl+Alt+T**
    - **Super+Left / Right**

Verification

```
gnome-extensions list --enabled
gsettings get org.gnome.desktop.interface color-scheme
```

Reflection

The best customization is not maximum customization. It is the minimum set of changes that improves comfort and workflow.

Lab 5: Add Printer, Scanner, Bluetooth Headset, and Second Monitor

Goal

Configure several common desktop peripherals successfully.

Skills covered

- CUPS
- printer queue
- scanner detection
- Bluetooth pairing
- display configuration

Why this lab matters

The Ubuntu Academy outline explicitly includes:

- CUPS printing
- scanners
- sound and Bluetooth pairing
- external displays and multi-monitor setups.

Tasks

1. Confirm CUPS is installed:

```
dpkg -l | grep cups
```

2. Open printer management:

```
system-config-printer &
```

3. Add a printer and print a test job:

```
lp /etc/hosts
lpstat -t
```

4. Plug in or detect a scanner:

```
lsusb
journalctl -f
```

5. Pair a Bluetooth headset.
6. Test sound output.
7. Connect a second monitor and arrange it in **Settings > Displays**.
8. Inspect outputs:

```
xrandr
```

Verification

- Printer shows as idle or ready
- Bluetooth headset appears as paired
- second monitor appears as connected
- sound test works

Reflection

> This lab helps learners see Ubuntu as a real-world desktop platform that handles office and classroom hardware, not just applications.

Lab 6: Configure Wi-Fi, Static IP, Firewall, and VPN

Goal

Configure a realistic desktop network profile and secure it.

Skills covered

- NetworkManager
- Wi-Fi
- static IP
- DNS
- UFW
- VPN

Why this lab matters

The Ubuntu Academy outline directly emphasizes:

- Wi-Fi and Ethernet
- static IP and DNS
- firewall with UFW
- VPN
- quick troubleshooting with `ping`, `ip`, and `nmcli`.

Tasks

1. Connect to Wi-Fi.
2. Check state:

```
nmcli device status
ip a
ip route show
```

3. Configure one connection profile with a static IP.
4. Set custom DNS.
5. Check `/etc/resolv.conf`.
6. Enable firewall:

```
sudo ufw status
sudo ufw enable
```

7. Allow one safe rule, for example SSH from local subnet.
8. Bring up a saved VPN profile if available.

Verification

```
ping -c 2 8.8.8.8
ping -c 2 archive.ubuntu.com
sudo ufw status
```

Reflection

This lab teaches that desktop networking is not just clicking a Wi-Fi icon. It is also understanding routes, names, trust, and host protection.

Lab 7: Create Users and Secure the Desktop

Goal

Create a second local user, manage privileges, and improve desktop security.

Skills covered

- adduser
- groups
- sudo
- password management
- session locking
- AppArmor status
- basic security habits

Why this lab matters

The Ubuntu Academy outline gives strong emphasis to:

- users and permissions
- groups and sudo
- strong passwords
- automatic locking
- AppArmor basics.

Tasks

1. Create a second user:

```
sudo adduser student2
```

2. Add the user to selected groups:

```
sudo adduser student2 sudo
```

3. Inspect current groups:

```
groups
```

4. Change a password with `passwd`.
5. Lock the current session:

```
loginctl lock-session
```

6. Check AppArmor status:

```
sudo apparmor_status
```

7. Review screen-lock settings in the GUI.

Verification

```
loginctl list-sessions
sudo apparmor_status | head
```

Reflection

This lab helps learners understand that real desktop security begins with people, accounts, and habits—not only with firewalls.

Lab 8: Backup, Restore, and Recover from Common Failures

Goal

Create a usable backup, perform a safe restore, and repair common package or session issues.

Skills covered

- Déjà Dup
- restore-first-to-safe-folder mindset
- `apt --fix-broken`
- `apt autoremove`
- routine recovery habits

Why this lab matters

The Ubuntu Academy outline explicitly includes:

- Déjà Dup
- Timeshift intro
- restore basics
- troubleshooting common issues
- cleanup and maintenance.

Tasks

1. Install or open Déjà Dup.
2. Configure a backup destination.
3. Run one backup.
4. Create a safe restore target:

```
mkdir -p ~/Restore-Test
```

5. Restore selected files into that target.
6. Simulate a package issue and repair it:

```
sudo apt --fix-broken install
sudo apt autoremove
```

7. Clean APT cache:

```
sudo apt clean
```

Verification

```
ls ~/Restore-Test
```

Reflection

A backup that has never been restored is only a hopeful assumption. This lab teaches recovery confidence.

Lab 9: Fix a Boot Issue Using Recovery Tools

Goal

Practice safe boot recovery using GRUB awareness, live USB rescue, and Boot-Repair.

Skills covered

- GRUB inspection
- live USB rescue
- Boot-Repair
- data-preservation-first mindset
- initramfs awareness

Why this lab matters

The Ubuntu Academy outline explicitly includes:

- GRUB2 recovery
- initramfs problems
- Boot-Repair tool
- live USB rescue
- saving user files before repair.

Ubuntu: The Complete Guide directly recommends Boot-Repair as the simplest first fix for many GRUB issues and advises backing up precious files before proceeding with deeper repair.

Tasks

1. Force the GRUB menu to appear.
2. Inspect an entry with **E**.
3. Boot from a live USB.
4. Identify disks:

```
lsblk -f
```

5. Mount the installed system:

```
sudo mkdir -p /mnt/rescue
sudo mount /dev/sda2 /mnt/rescue
ls /mnt/rescue/home
```

6. Copy user files to an external target if needed.
7. Install Boot-Repair in the live session:

```
sudo add-apt-repository ppa:yannubuntu/boot-repair
sudo apt-get update
sudo apt-get install boot-repair
```

```
boot-repair
```

8. Try **Recommended repair** first.

Verification

- Boot-Repair completes
- user files are copied or verified safe
- Ubuntu boots again or at least reaches a healthier recovery state

Reflection

> This lab teaches the most important recovery lesson in the book: **preserve data first, repair second**.

Lab 10: Build Your Own Ubuntu Desktop Checklist

Goal

Create a reusable personal checklist for installing, configuring, securing, maintaining, and recovering an Ubuntu Desktop system.

Skills covered

- summarization
- self-documentation
- operational discipline
- role-based deployment thinking

Why this lab matters

The Ubuntu Academy outline repeatedly emphasizes practical operations, best practices, and hands-on learning. A final checklist lab helps turn scattered skills into a repeatable method.

Tasks

Create your own checklist with sections such as:

- installation
- first boot
- updates
- apps
- GNOME customization
- users and passwords
- networking
- firewall
- peripherals
- backups
- recovery media
- community/help resources

CLI example: create a checklist folder and file

```
mkdir -p ~/Documents/ubuntu-notes
nano ~/Documents/ubuntu-notes/my-ubuntu-desktop-checklist.txt
```

Example checklist starter

```
[ ] Install Ubuntu Desktop
[ ] Run apt update && apt upgrade -y
[ ] Install LibreOffice, Firefox tools, VLC
[ ] Configure Wi-Fi and test ping
[ ] Enable UFW
[ ] Check apparmor_status
[ ] Pair Bluetooth headset
[ ] Add printer
[ ] Configure backup
[ ] Test restore
[ ] Prepare live USB rescue media
```

Verification

```
cat ~/Documents/ubuntu-notes/my-ubuntu-desktop-checklist.txt
```

Reflection

This final lab is important because professionals do not rely only on memory. They build procedures. A checklist turns Ubuntu knowledge into a repeatable operational habit.

Chapter Reflection

Guided Hands-On Labs turns Ubuntu Desktop learning into practice. The Ubuntu Academy companion outline explicitly expects scenario-based tasks, and the reference books strongly support this style through VirtualBox installation, package management, GNOME workflows, hardware setup, networking, security, backup, and boot repair. These labs are designed to move from safe installation and workstation setup, through software management and customization, into peripherals, networking, security, backup, and recovery. By the time readers complete all ten labs, they will not only know Ubuntu Desktop conceptually—they will have *used* it in the kinds of real situations that matter most: building, maintaining, recovering, and documenting a working desktop environment.

Key Takeaways

- **Ubuntu Desktop skills become real only when practiced in realistic scenarios.**
- **VM-based labs are excellent first environments** for safe Ubuntu learning.
- **Package management should be practiced across APT, Snap, Flatpak, and `.deb`.**
- **Peripheral, network, and security labs** are essential because desktop use is more than launching apps.
- **Backup and boot-repair labs matter because recovery is part of real desktop administration.**
- **A personal checklist is one of the best final outputs** of Ubuntu training because it turns knowledge into repeatable practice.

Appendix A — Ubuntu Desktop Quick Reference

This appendix is designed as a **fast practical cheat sheet** for everyday Ubuntu Desktop work. The uploaded Ubuntu Academy outline explicitly calls for a **Quick Reference Cheat Sheet** covering **package commands, networking, hardware queries, and desktop quick commands**, while the attached Ubuntu references provide the command patterns used throughout earlier chapters.

A useful way to read this appendix is to treat it like a **desktop field card**. It is not meant to replace the full explanations in earlier chapters. It is meant to give readers a compact, copy-friendly reminder of the commands they are most likely to need in daily use, classroom labs, and exam-style practice. The attached references especially support this approach through repeated use of APT, Snap, Flatpak, hardware detection, man pages, process tools, and desktop support workflows.

1. Essential Commands for APT, Snap, and Flatpak

Ubuntu Academy explicitly places **APT, Snap, and Flatpak** at the center of package-management skills, including `apt update`, `apt upgrade`, `apt install`, `apt remove`, `apt search`, `snap install/search/remove/refresh/list`, and Flatpak integration with **Flathub**. The attached Ubuntu Linux Bible and Mastering Ubuntu references both show practical examples of these commands in normal desktop use.

APT quick commands

```
sudo apt update
sudo apt upgrade -y
sudo apt install <package>
sudo apt remove <package>
apt search <keyword>
apt list --upgradable
sudo apt autoremove
sudo apt clean
```

What they do:

- `sudo apt update` → refresh package metadata from repositories.
- `sudo apt upgrade -y` → install available upgrades for installed packages.
- `sudo apt install <package>` → install a package from configured repositories.
- `sudo apt remove <package>` → remove an installed package.
- `apt search <keyword>` → search repositories for matching packages.
- `apt list --upgradable` → list packages with available updates.
- `sudo apt autoremove` → remove no-longer-needed dependency packages.
- `sudo apt clean` → clear cached package archives.

.deb quick commands

Ubuntu Academy explicitly includes **`dpkg` install/remove** and **gdebi for GUI**.

```
sudo dpkg -i <file>.deb
sudo dpkg -r <package>
dpkg -l | head
```

- `sudo dpkg -i <file>.deb` → install a local Debian package file.
- `sudo dpkg -r <package>` → remove a package installed through Debian packaging.
- `dpkg -l | head` → list installed packages.

Snap quick commands

The Ubuntu Academy outline explicitly lists **snap install/search/remove/refresh/list**, and the Ubuntu Linux Bible shows concrete examples such as `snap install kolourpaint` and `snap list`.

```
snap find <keyword>
sudo snap install <package>
sudo snap remove <package>
sudo snap refresh
snap list
```

- `snap find <keyword>` → search Snap packages.
- `sudo snap install <package>` → install a Snap.
- `sudo snap remove <package>` → remove a Snap.
- `sudo snap refresh` → update installed Snaps.
- `snap list` → show installed Snaps.

Flatpak quick commands

Ubuntu Academy explicitly includes **installing Flatpak, adding Flathub, and using flatpak install/run/update**. The Ubuntu Linux Bible provides a concrete example with **Foliate** from Flathub.

```
sudo apt install flatpak
flatpak remote-add --if-not-exists flathub
https://flathub.org/repo/flathub.flatpakrepo
flatpak install flathub <app-id>
flatpak run <app-id>
flatpak update
flatpak list
```

- `sudo apt install flatpak` → enable Flatpak support.
- `flatpak remote-add --if-not-exists flathub ...` → add Flathub as the application source.
- `flatpak install flathub <app-id>` → install a Flatpak app.
- `flatpak run <app-id>` → launch a Flatpak app from the command line.
- `flatpak update` → update installed Flatpak apps.
- `flatpak list` → show installed Flatpaks.

2. Networking Quick Commands

Ubuntu Academy explicitly calls out **NetworkManager basics, Wi-Fi/Ethernet, static IP, DNS, VPN, hostname, and diagnostics using `ping`, `ip`, and `nmcli`**.

A practical desktop networking quick set is:

```
nmcli device status
nmcli connection show
nmcli device wifi list
ip a
ip route show
ping -c 2 8.8.8.8
ping -c 2 archive.ubuntu.com
hostname
hostnamectl
cat /etc/resolv.conf
```

What they do:

- `nmcli device status` → show NetworkManager's view of device state.
- `nmcli connection show` → list saved connection profiles.
- `nmcli device wifi list` → show visible Wi-Fi networks.
- `ip a` → show network interfaces and assigned addresses.
- `ip route show` → show routing and default gateway info.
- `ping -c 2 8.8.8.8` → test raw internet connectivity.
- `ping -c 2 archive.ubuntu.com` → test connectivity plus DNS resolution.
- `hostname` / `hostnamectl` → show the current system name and identity.
- `cat /etc/resolv.conf` → inspect DNS resolver configuration.

Static IP / Wi-Fi / hotspot quick examples

```
nmcli device wifi connect "SSID" password "password"

sudo nmcli connection modify "Wired 1" ipv4.method manual
ipv4.addresses 192.168.1.50/24 ipv4.gateway 192.168.1.1 ipv4.dns
"1.1.1.1 8.8.8.8"

sudo nmcli connection up "Wired 1"

nmcli device wifi hotspot ifname wlp2s0 ssid UbuntuHotspot
password "Ubuntu12345"
```

These match the Academy outline's emphasis on Wi-Fi setup, static IP, DNS, and hotspot sharing.

Firewall quick commands

Ubuntu Academy explicitly includes **`ufw enable/status` and allow/deny rules**.

```
sudo ufw status
sudo ufw enable
sudo ufw allow 22/tcp
sudo ufw deny 22/tcp
```

3. Hardware Information Commands

Ubuntu Academy explicitly lists **`lspci`, `lsusb`, `lshw`, and `inxi`** under hardware detection, plus graphics, audio, Bluetooth, and external-display troubleshooting. The Ubuntu Linux Bible strongly supports `lspci` and `lsusb` for identifying malfunctioning devices by model.

```
lspci
lspci -vvv
lsusb
sudo lshw -short
inxi -F
lsblk -f
xrandr
aplay -l
upower -e
```

What they do:

- `lspci` → list PCI devices such as graphics, audio, Ethernet, and controllers.
- `lspci -vvv` → show more verbose hardware details.
- `lsusb` → list USB devices.
- `sudo lshw -short` → show a summarized hardware tree.
- `inxi -F` → give a readable full system summary.
- `lsblk -f` → show storage devices, filesystems, and mount points.
- `xrandr` → inspect connected display outputs and resolutions.
- `aplay -l` → list audio playback hardware.
- `upower -e` → list power and battery devices.

Graphics / driver quick commands

```
lspci | grep -i vga
sudo ubuntu-drivers devices
nvidia-smi
sudo prime-select query
```

These align with the Academy outline's focus on proprietary vs open drivers and PRIME for hybrid graphics.

4. User and Permission Quick Commands

Ubuntu Academy explicitly includes **`adduser`**, **`usermod`**, **`passwd`**, **`deluser`**, **groups, sudo, chmod, and chown`** in desktop context.

```
whoami
groups
sudo adduser <user>
sudo usermod -aG <group> <user>
sudo adduser <user> sudo
passwd
sudo passwd <user>
sudo deluser <user>
ls -l
chmod +x <file>
sudo chown <user>:<group> <file>
loginctl list-sessions
loginctl lock-session
```

What they do:

- `whoami` → show the current user.
- `groups` → show the user's group memberships.
- `sudo adduser <user>` → create a new user account.
- `sudo usermod -aG <group> <user>` → append a user to a group.
- `sudo adduser <user> sudo` → grant admin-capable group membership.
- `passwd` → change your password.
- `sudo passwd <user>` → reset another user's password as admin.
- `sudo deluser <user>` → remove a user account.
- `ls -l` → show permissions, ownership, and file details.
- `chmod +x <file>` → make a file executable.
- `sudo chown <user>:<group> <file>` → change ownership.
- `loginctl list-sessions` → list user sessions.
- `loginctl lock-session` → lock the current session.

5. Update and Troubleshooting Quick Commands

Ubuntu Academy explicitly emphasizes **manual and automatic updates, troubleshooting frozen apps, dependency handling, slow boot, driver conflicts, and common desktop issues**. The attached Ubuntu references support this with commands for logs, process management, package repair, and system diagnostics.

System update and repair

```
sudo apt update
sudo apt upgrade -y
apt list --upgradable
sudo apt --fix-broken install
sudo dpkg-reconfigure <package>
sudo apt autoremove
sudo apt clean
```

- `sudo apt --fix-broken install` → repair dependency problems after interrupted or broken installs.
- `sudo dpkg-reconfigure <package>` → rerun package configuration.

Logs and status

```
journalctl -b -n 20
journalctl -f
dmesg | tail
df -h
free -h
uptime
hostnamectl
```

What they do:

- `journalctl -b -n 20` → show recent log entries from the current boot.
- `journalctl -f` → follow logs live.
- `dmesg | tail` → show recent kernel messages.
- `df -h` → check free disk space.
- `free -h` → check memory use.
- `uptime` → check system uptime and load.
- `hostnamectl` → inspect host identity and platform info.

Process and performance quick commands

```
top
ps -eo pid,user,group,gid,vsz,rss,comm --sort=-vsz | head
gnome-system-monitor &
```

These are directly supported by the Ubuntu Linux Bible's process-management and graphical administration workflows.

Boot / recovery quick commands

Ubuntu Academy explicitly includes **GRUB, initramfs, and Boot-Repair** in installation and troubleshooting domains. *Ubuntu: The Complete Guide* explicitly recommends **Boot-Repair** for many GRUB problems.

```
sudo update-initramfs -u
sudo update-grub
lsblk -f
sudo mount /dev/sda2 /mnt/rescue
```

These are useful in repair or live-USB rescue contexts.

Compact One-Page Cheat List

For readers who want a single compact block, this is the shortest practical version:

```
# APT
sudo apt update
sudo apt upgrade -y
sudo apt install <package>
sudo apt remove <package>
apt search <keyword>
apt list --upgradable
sudo apt --fix-broken install
sudo apt autoremove
sudo apt clean

# Snap
snap find <keyword>
sudo snap install <package>
sudo snap remove <package>
sudo snap refresh
snap list

# Flatpak
sudo apt install flatpak
flatpak remote-add --if-not-exists flathub
https://flathub.org/repo/flathub.flatpakrepo
flatpak install flathub <app-id>
flatpak run <app-id>
flatpak update
flatpak list

# Networking
nmcli device status
nmcli connection show
nmcli device wifi list
ip a
ip route show
ping -c 2 8.8.8.8
ping -c 2 archive.ubuntu.com
hostnamectl
cat /etc/resolv.conf
sudo ufw status
sudo ufw enable

# Hardware
lspci
lsusb
sudo lshw -short
inxi -F
lsblk -f
xrandr
aplay -l
upower -e
```

```
# Users and permissions
whoami
groups
sudo adduser <user>
sudo adduser <user> sudo
passwd
sudo passwd <user>
ls -l
chmod +x <file>
sudo chown <user>:<group> <file>
loginctl lock-session

# Troubleshooting
journalctl -b -n 20
journalctl -f
dmesg | tail
df -h
free -h
uptime
top
gnome-system-monitor &
sudo update-initramfs -u
sudo update-grub
```

This cheat list is directly aligned with the Appendix A quick-reference expectation in the Ubuntu Academy outline and with the command patterns demonstrated across the attached Ubuntu references.

Appendix B — GUI Path Reference

This appendix is a **desktop-side navigation map** for Ubuntu users who prefer to remember **where to click** before memorizing commands. The uploaded Ubuntu Academy outline explicitly calls for a GUI-aware companion covering **Settings > Appearance**, **Ubuntu Software / App Center**, **Settings GUI for users**, **GUI networking**, **CUPS printer setup**, **sound and Bluetooth pairing**, **backup tools**, and **startup/session management**.

The attached books reinforce this approach. *Mastering Ubuntu* explains that users start applications from **Activities**, the **Dash/Dock**, and the **app grid**, and it identifies **Software Updater** and **Ubuntu Software** as standard desktop entry points for updates and application management. *Ubuntu Linux Bible* explains that Ubuntu Software can be opened from the Dock and used to browse or search for applications. *Ubuntu: The Complete Guide* adds practical GUI paths for **Sound Settings**, **Bluetooth**, **Startup Applications**, and other daily desktop operations.

A useful way to read this appendix is to imagine it as a **campus map** for Ubuntu Desktop. It tells readers which building to enter, which hallway to take, and which room usually contains the control they need.

1. Settings Menu Map

The Ubuntu Academy outline explicitly positions the **Settings app** as one of the most important desktop tools, alongside Nautilus/Files and Extensions. It also names major Settings areas such as **Appearance**, **Network**, **Users**, **Power**, **Displays**, **Sound**, **Bluetooth**, and backup-related desktop tools in surrounding chapters.

How to open Settings

Common GUI paths:

- **Top-right system menu** → click system/status area → **Settings**
- **Activities** → type **Settings**
- **Show Applications** → **Settings** icon

This general launch pattern is grounded in *Mastering Ubuntu*, which explains that users open applications from the **Activities overview**, the **Dash**, and the **app grid**, and that searching starts immediately when typing an application name.

Core Settings map

Below is a practical Ubuntu Desktop map of the most useful sections.

Appearance

Settings → Appearance

Use this path for:

- wallpaper
- light/dark mode
- basic visual style

The Ubuntu Academy outline explicitly lists **Themes, icons, wallpapers (Settings > Appearance)** as a core customization area.

Network

Settings → Network
Settings → Wi-Fi

Use these paths for:

- Ethernet status
- Wi-Fi network selection
- connection details
- IPv4/IPv6 settings
- proxy access in related network workflows

The Ubuntu Academy outline explicitly calls out **NetworkManager basics: GUI connections, Wi-Fi/Ethernet setup**, plus **static IP, DNS, proxy settings**.

Bluetooth

Settings → Bluetooth

Use this path for:

- enabling Bluetooth
- making the desktop discoverable
- pairing speakers, headsets, keyboards, or phones

Ubuntu: The Complete Guide explicitly describes using the Bluetooth application/settings window, enabling the top switches, searching for devices with the **+** button, and then selecting the Bluetooth device in **Sound Settings** once paired.

Sound

Settings → Sound

Use this path for:

- selecting output device
- selecting input device
- testing speakers
- switching to a Bluetooth speaker or headset after pairing

Ubuntu: The Complete Guide explicitly states that once the **Sound Settings** menu is open, users can click **Test Sound**, choose devices from the left pane, and select Bluetooth audio devices in the Output tab.

Displays

Settings → Displays

Use this path for:

- monitor arrangement
- display mirroring vs extended desktop
- scale and resolution
- external monitor handling

The Ubuntu Academy outline explicitly includes **external displays, docking, and multi-monitor setups**, making Displays one of the key GUI destinations.

Power

Settings → Power

Use this path for:

- battery-related behavior
- suspend/screen blanking
- laptop power preferences

The Ubuntu Academy outline explicitly names **Power management: Battery settings, TLP for laptops**.

Users

Settings → Users

Use this path for:

- adding or managing users
- changing user type
- password change workflows
- administrator vs standard user behavior

The Ubuntu Academy outline explicitly lists **Creating/managing users: Settings GUI**.

Sharing

Settings → Sharing

Use this path for:

- remote desktop / screen sharing
- sharing-related desktop connectivity

The Ubuntu Academy outline's networking and sharing scope supports this as a desktop-control area, especially for screen-sharing and remote access use cases.

About

Settings → About

Use this path for:

- device name / hostname
- Ubuntu version
- hardware overview
- system identity during support tasks

The Ubuntu Academy outline explicitly includes **basic hostname and domain setup** and broader desktop support workflows, which makes **About** a useful GUI checkpoint.

2. App Center Quick Paths

The Ubuntu Academy outline explicitly includes **Ubuntu Software / App Center: GUI installation, updates, Snap/DEB preferences** under package management. *Mastering Ubuntu* explains that **Ubuntu Software** is the place to search, download, install, and remove software without opening a browser, and that it is accessible from both the launcher and the app drawer. *Ubuntu Linux Bible* also says Ubuntu Software can be opened from the Dock and used to browse by category or search by name.

How to open App Center / Ubuntu Software

Common GUI paths:

- **Dock / Launcher** → **Ubuntu Software** or **App Center**
- **Show Applications** → **Ubuntu Software** / **App Center**
- **Activities** → type **Software** or **App Center**

Quick task map inside App Center

Search for an application

App Center → Search box → type application name

Mastering Ubuntu explicitly says users can search, download, install, and remove software in a single window, and that Firefox can be found by entering its name in the search bar.

Browse by category

App Center → Categories → choose category (for example Audio, Players, Graphics, Utilities)

Ubuntu: The Complete Guide explicitly mentions opening the **Software Centre**, clicking **audio** under **Categories**, and then choosing **Players** to see music players.

Install an application

App Center → Search or browse → open app page → Install

This flow is directly supported by *Mastering Ubuntu*, which describes clicking an application in Ubuntu Software and using the **Install** button.

Remove an application

App Center → Installed / app details → Remove

The same reference notes that Ubuntu Software can also **remove** applications, not only install them.

Review software source / packaging context

App Center → app page → source label / packaging details

The Ubuntu Academy outline explicitly calls for **Snap/DEB preferences** in the GUI software context, so readers should look for the app-source indicator on an app's detail page.

Install Flatpak support through the GUI ecosystem

App Center / Software Center → search plugin/backend package if needed

Mastering Ubuntu explicitly explains that Flatpak can be integrated with the **GNOME Software Center** by installing the Flatpak plug-in package.

3. Common Admin Tasks by GUI Path

This section maps common Ubuntu Desktop administrative tasks to the GUI locations where users are most likely to perform them. The Ubuntu Academy outline explicitly supports this style by listing many tasks in both GUI and CLI form, including **Software Updater vs command-line**, **Settings GUI**, **CUPS setup**, **sound and Bluetooth**, **networking**, **users**, and **backup/restore basics**.

Update the system

Activities / Show Applications → Software Updater

Mastering Ubuntu explicitly states that users should use **Software Updater** to keep the computer safe, secure, and up to date, and that it regularly installs security updates and critical bug fixes.

Install software

Dock / Show Applications → Ubuntu Software / App Center

Supported by both *Mastering Ubuntu* and *Ubuntu Linux Bible*.

Manage printer settings

Settings → Printers
or
Activities → search “Printers” / “Printer Settings”

The Ubuntu Academy outline explicitly includes **Printers and scanners: CUPS setup, adding devices**, and earlier book-aligned material ties this to GUI printer administration.

Pair a Bluetooth device

Settings → Bluetooth
then
Settings → Sound to choose the audio output if it is a speaker/headset

This exact pairing flow is described in *Ubuntu: The Complete Guide*.

Troubleshoot sound by GUI

Settings → Sound → Test Sound / select output
Optional advanced tool from GUI software source:
App Center / Software Centre → install PulseAudio Volume Control

Ubuntu: The Complete Guide explicitly says to check the correct output device in **Sound Settings**, use **Test Sound**, and install **PulseAudio Volume Control** from the software centre for finer control.

Add or manage users

Settings → Users

Supported directly by the Ubuntu Academy outline.

Manage startup apps

Activities → search “Startup Applications”
or launch the Startup Applications tool

Ubuntu: The Complete Guide explicitly describes opening **Startup Applications** from the Dash/search and using it to control login-time programs. This was covered earlier from the same reference base.

Change display arrangement

Settings → Displays

Supported by the Academy outline’s external-display and monitor-management coverage.

Change network settings

Settings → Wi-Fi
Settings → Network
Settings → Network Proxy (or the proxy section within the network settings model, depending on release)

The Academy outline explicitly includes **GUI connections, static IP, DNS, proxy settings**.

Backup by GUI

Show Applications / Activities → Déjà Dup / Backups

The Academy outline explicitly includes **Backup and restore basics: Déjà Dup** under system maintenance.

Inspect help

Help menu in an app → Contents
or
Activities → Help

Ubuntu: The Complete Guide explicitly says many apps provide built-in help through **Help → Contents**, and that **F1** often opens help.

4. GNOME Customization Quick Map

The Ubuntu Academy outline explicitly places GNOME customization under:

- **Settings > Appearance**
- **GNOME Extensions**
- **gnome-tweaks**
- dock/panel adjustments
- keyboard shortcuts
- fonts
- advanced GSettings/dconf paths.

The attached books support this overall map by describing GNOME as Ubuntu's default desktop, explaining how apps are launched from Activities and the Dash, and pointing to extensions and GUI customization paths as part of daily desktop work.

Wallpaper and light/dark mode

Settings → Appearance

This path is named directly in the Ubuntu Academy outline.

Themes, icons, cursor, and deeper appearance options

Activities / Show Applications → GNOME Tweaks → Appearance

The Ubuntu Academy outline explicitly names **gnome-tweaks** and themes/icons/fonts in the customization section.

Extensions

Activities / Show Applications → Extensions
or
GNOME Tweaks → Extensions
or
browser-based **extensions.gnome.org** workflow as noted in the outline

The Academy outline explicitly names **extensions.gnome.org** and **gnome-tweaks**.

Keyboard shortcuts

Settings → Keyboard
or the shortcuts section inside keyboard/system settings depending on release

The outline explicitly includes **keyboard shortcuts** as part of desktop customization.

Dock / panel behavior

Settings → **Appearance** for basic visual behavior
and/or
GNOME Tweaks / Extensions for deeper dock-related changes

The Academy outline explicitly includes **Dock/panel adjustments**.

Fonts

GNOME Tweaks → **Fonts**

The outline explicitly includes **fonts** in customization.

Advanced configuration

GNOME Tweaks first
then deeper configuration via **GSettings / dconf** when necessary

The Academy outline explicitly frames **GSettings / dconf** as the advanced layer after the regular GUI tools.

A practical GNOME customization rule, supported by the broader Ubuntu Desktop material, is to start with **Settings**, then move to **GNOME Tweaks**, then use **Extensions**, and only then consider more advanced direct configuration.

5. Quick GUI Task Table

Below is a compact map readers can use quickly during labs.

Task	GUI Path
Open system settings	**Top-right menu → Settings or Activities → Settings**
Change wallpaper / dark mode	**Settings → Appearance**
Connect to Wi-Fi	**Top-right menu → Wi-Fi or Settings → Wi-Fi**
Configure Ethernet / IP / DNS	**Settings → Network**
Pair Bluetooth device	**Settings → Bluetooth**
Select audio output / test sound	**Settings → Sound**
Arrange monitors	**Settings → Displays**
Manage users	**Settings → Users**
Install software	**Ubuntu Software / App Center**
Run desktop updates	**Software Updater**
Manage startup apps	**Startup Applications**
Manage printers	**Settings → Printers or Printer Settings**
Open help	**Help app or Help → Contents inside applications**
Tune deeper GNOME appearance	**GNOME Tweaks**
Manage extensions	**Extensions or GNOME Tweaks → Extensions**
Start backup setup	**Déjà Dup / Backups**

Appendix C — Troubleshooting Flowcharts

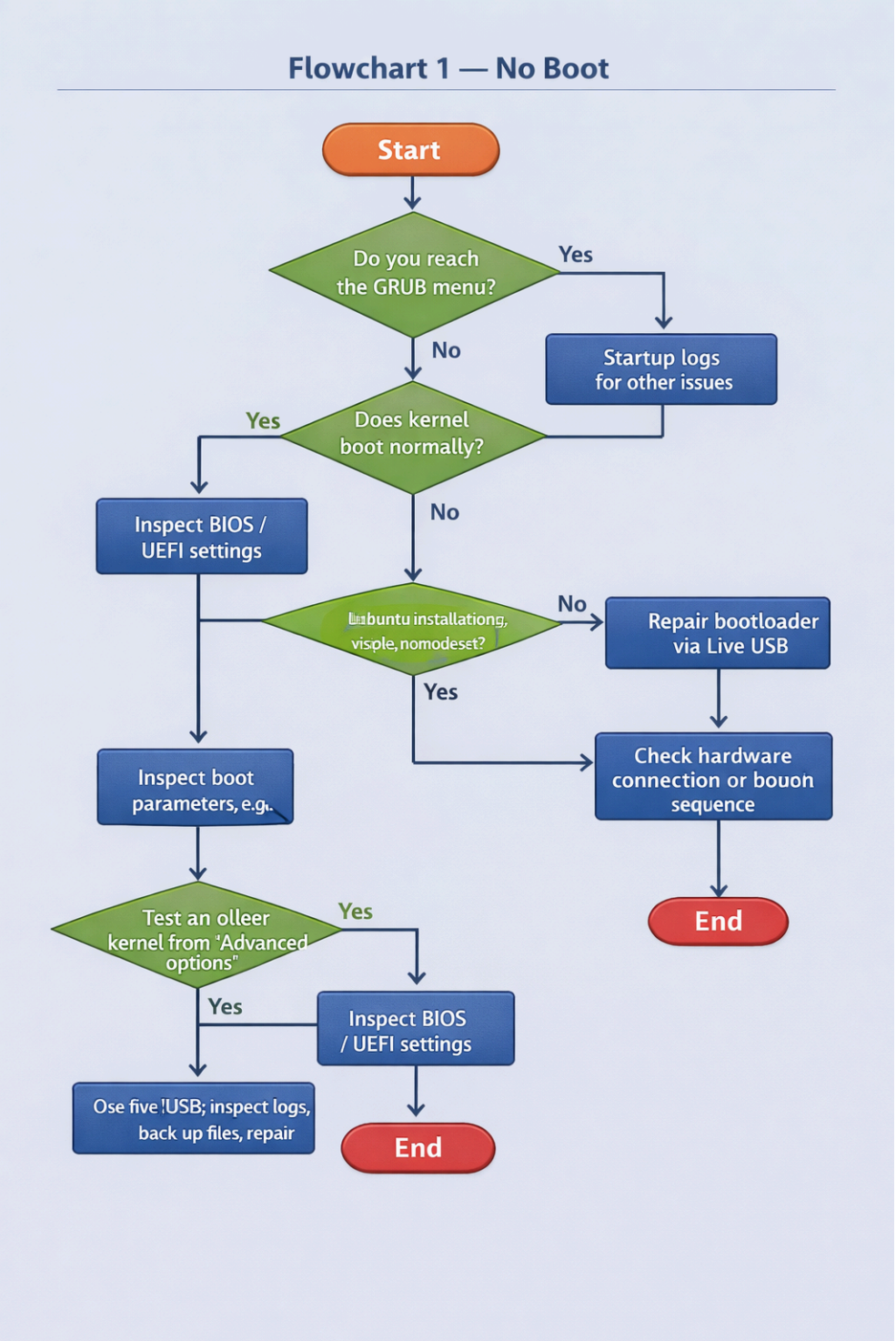
This appendix turns common Ubuntu Desktop problems into **simple decision paths**. The uploaded Ubuntu Academy outline explicitly emphasizes **frozen apps, driver conflicts, slow boot, package issues, Wi-Fi/firewall troubleshooting, GRUB/initramfs recovery, and printer/scanner setup**, so a flowchart appendix is a natural fit for the book.

The attached references support these flowcharts well: *Ubuntu: The Complete Guide* explains startup failures in **pre-GRUB, GRUB, and post-GRUB** stages and recommends **Boot-Repair** plus live rescue workflows; *Ubuntu Linux Bible* supports hardware detection with `dmesg`, `journalctl`, `lsusb`, service checks, and package repair tools; and *Mastering Ubuntu* reinforces desktop package workflows and software-center access.

A useful way to read these is: **start at the top, answer each question honestly, and only move to the next branch when the previous one is checked**.

Flowchart 1 — No Boot

Ubuntu: The Complete Guide explicitly recommends splitting startup failures into **pre-GRUB, GRUB, and post-GRUB** stages, and says Boot-Repair is often the simplest first fix for GRUB-related failures. It also advises using rescue media to verify disks and back up important files before deeper repair.

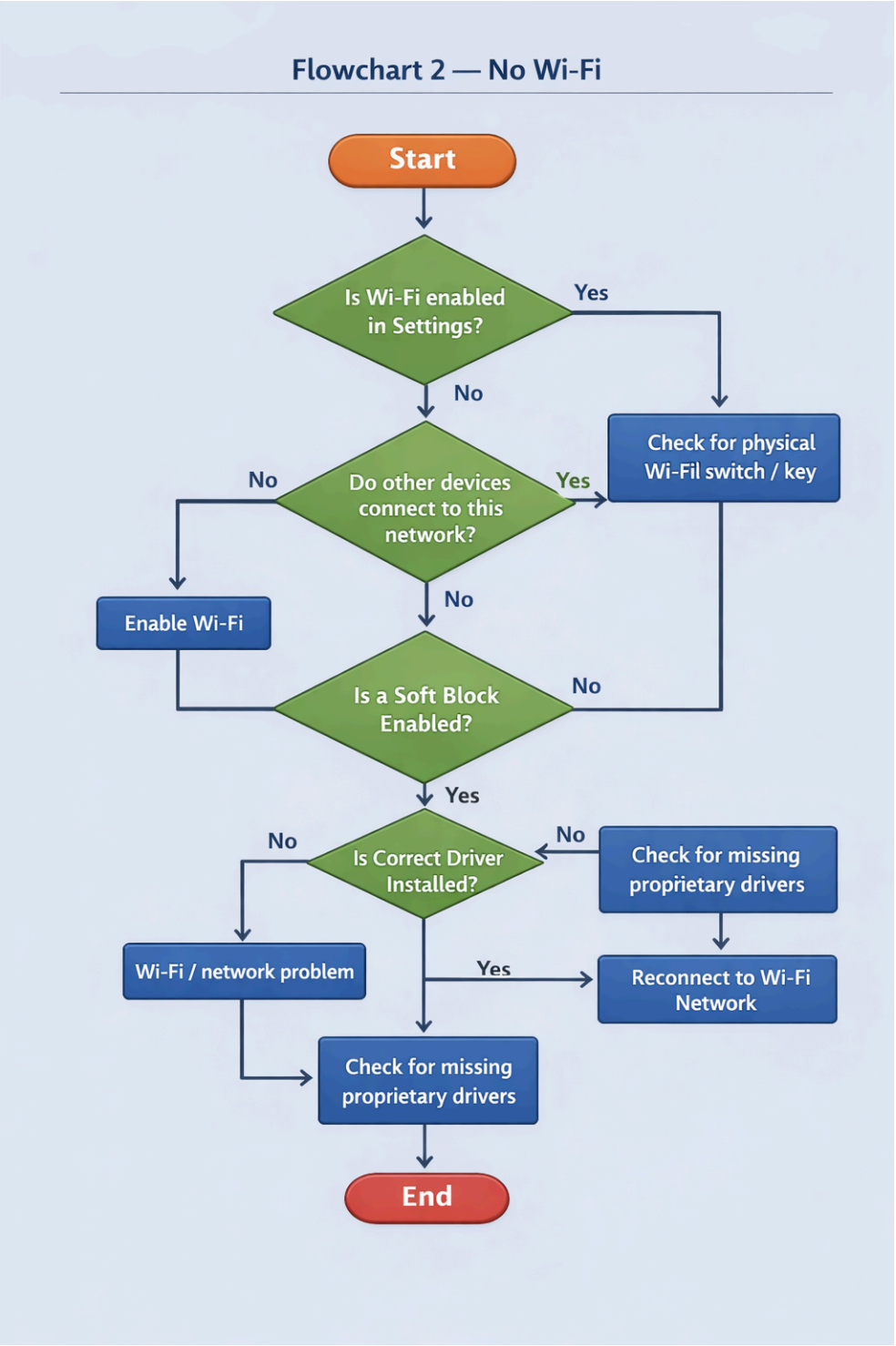


Quick commands for rescue context

```
lsblk -f
sudo mkdir -p /mnt/rescue
sudo mount /dev/sda2 /mnt/rescue
ls /mnt/rescue/home
sudo update-initramfs -u
sudo update-grub
```

Flowchart 2 — No Wi-Fi

The Ubuntu Academy outline explicitly includes **NetworkManager, Wi-Fi/Ethernet setup, static IP, DNS, proxy, and troubleshooting with ping and nmcli**. The attached references support using NetworkManager and the standard desktop network tools for diagnosis.

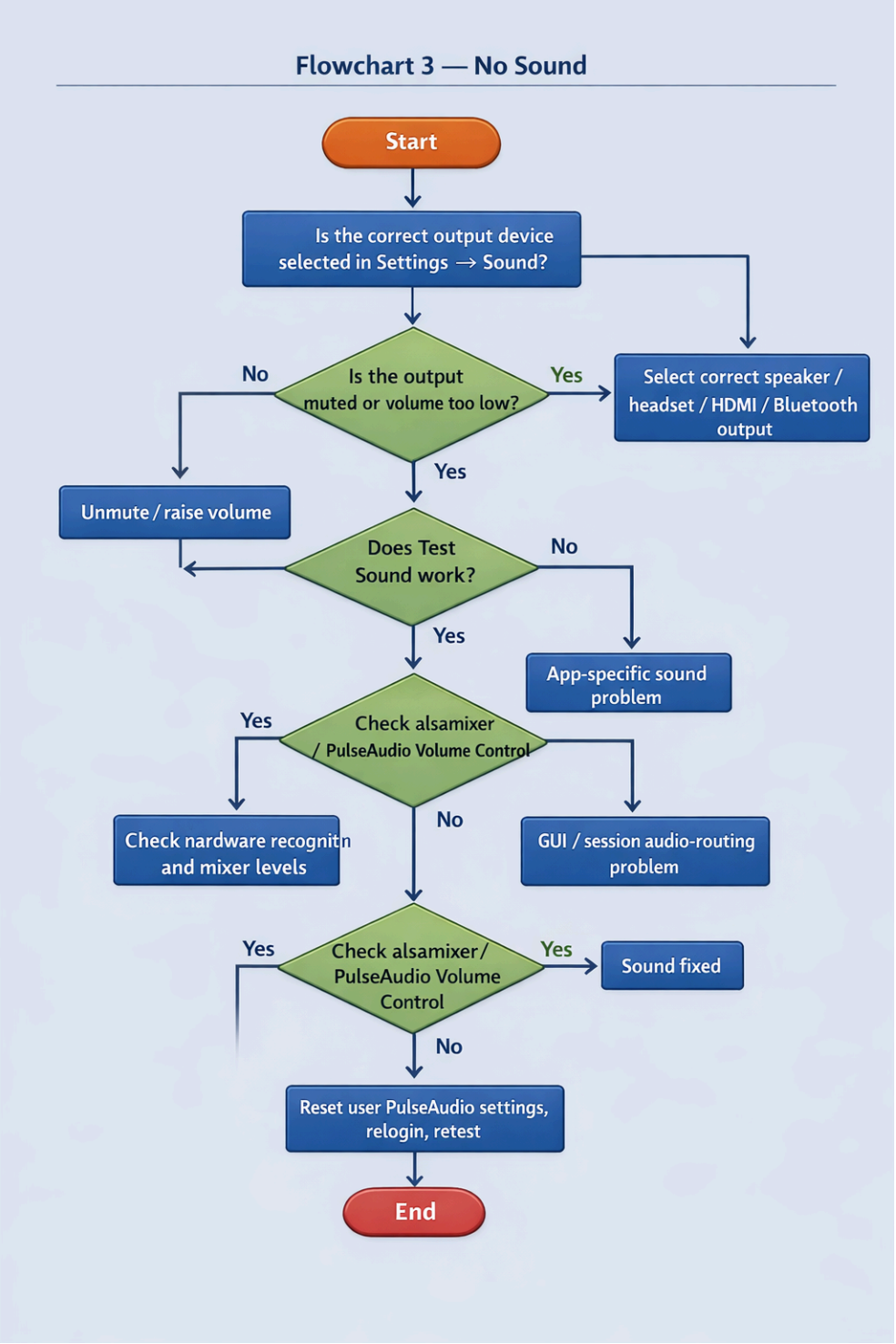


Quick commands

```
nmcli device status
nmcli device wifi list
nmcli connection show
ip a
ip route show
ping -c 2 192.168.1.1
ping -c 2 8.8.8.8
ping -c 2 archive.ubuntu.com
cat /etc/resolv.conf
```

Flowchart 3 — No Sound

Ubuntu: The Complete Guide explicitly recommends **Sound Settings**, **Test Sound**, `aplay /usr/share/sounds/alsa/Front_Center.wav`, `alsamixer`, and, when needed, resetting user PulseAudio settings.

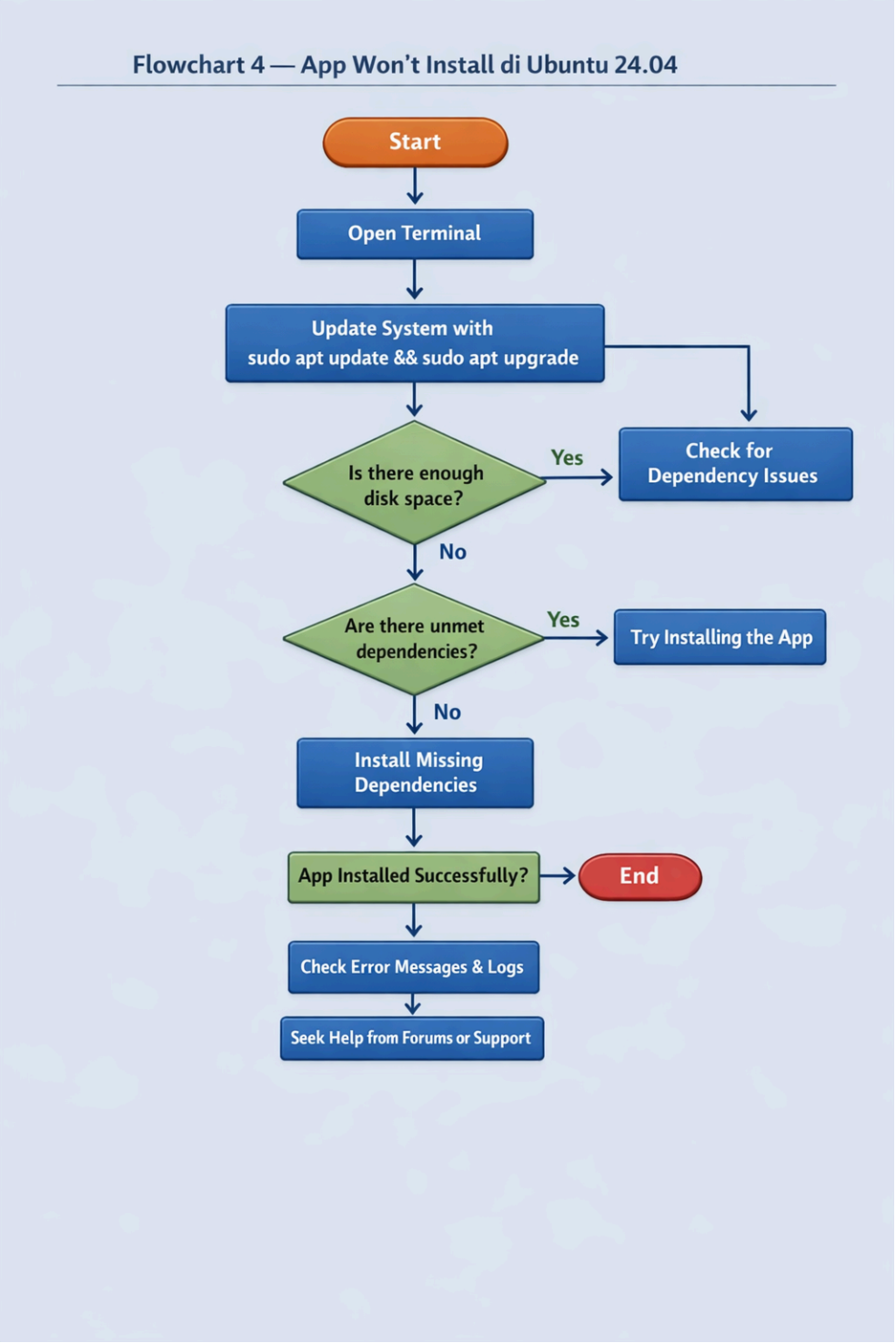


Quick commands

```
aplay /usr/share/sounds/alsa/Front_Center.wav
alsamixer
aplay -l
sudo rm -r ~/.config/pulse; pulseaudio -k
```

Flowchart 4 — App Won’t Install

The Ubuntu Linux Bible explicitly supports this area with `apt update`, `apt --fix-broken install`, `dpkg-reconfigure`, and repository-awareness. It also explains that “Unable to locate package” often means outdated metadata or a wrong package name.

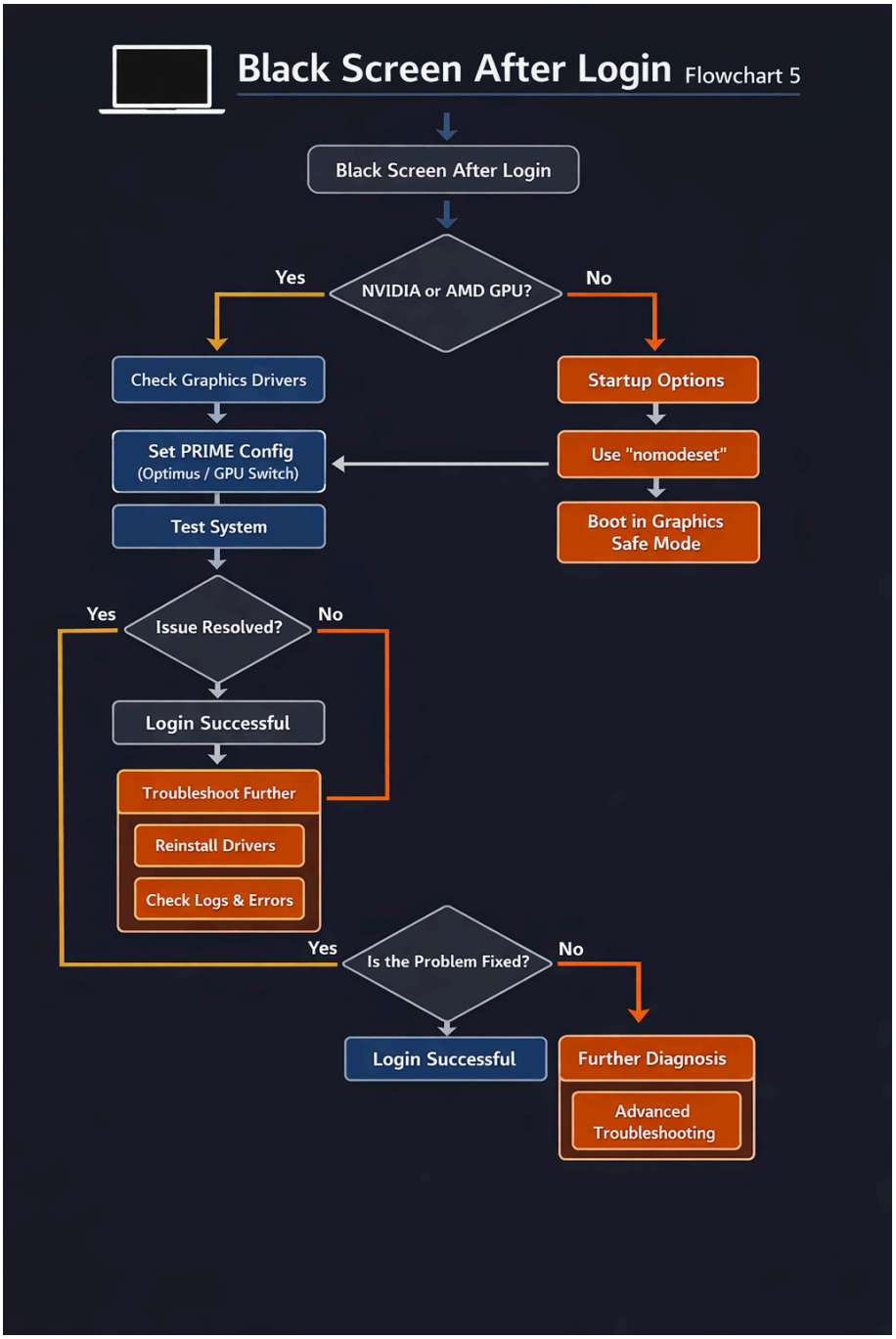


Quick commands

```
sudo apt update
apt search <keyword>
sudo apt install <package>
sudo apt --fix-broken install
sudo dpkg -i <file>.deb
sudo dpkg-reconfigure <package>
ls /etc/apt/sources.list.d/
```

Flowchart 5 — Black Screen After Login

The uploaded Ubuntu Academy outline explicitly includes **graphics driver conflicts**, **PRIME**, and **troubleshooting common issues**. *Ubuntu: The Complete Guide* also points to boot-stage thinking and graphics-safe boot options such as `nomodeset` for startup-related display issues.



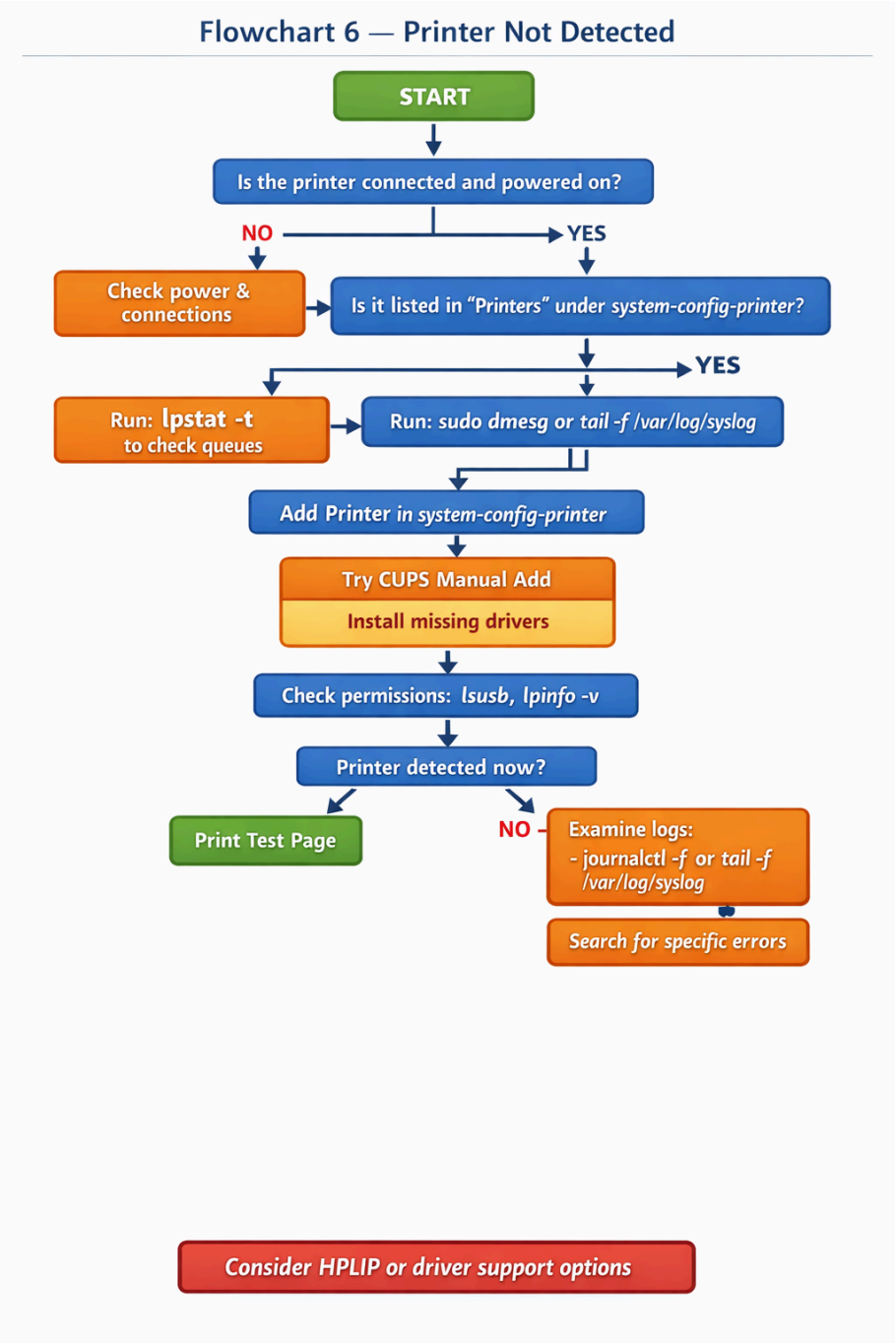
Quick commands

```
lspci | grep -i vga
sudo ubuntu-drivers devices
sudo prime-select query
gnome-extensions list --enabled
gnome-extensions disable <extension-id>
journalctl -b -n 50
```

If boot graphics are implicated, *Ubuntu: The Complete Guide* specifically notes that temporary boot parameters such as `nomodeset` can help in some graphics-related startup situations.

Flowchart 6 — Printer Not Detected

The Ubuntu Academy outline explicitly includes **CUPS setup, adding devices, and printers/scanners**. The Ubuntu Linux Bible explains that Ubuntu printing is built around **CUPS**, that `system-config-printer` is the graphical tool, and that `lpstat -t` is useful for queue and status checks. It also explains that Linux device detection is dynamic through the kernel and Udev, and that `dmesg`, `journalctl -f`, or `tail -f /var/log/syslog` can help when plugging devices in.



Quick commands

```
lsusb
systemctl status cups
lpstat -t
system-config-printer &
journalctl -f
dmesg | tail
```

Optional print test

```
lp /etc/hosts
```

This exact `lp /etc/hosts` style is used in the Ubuntu Linux Bible's printer exercises.

One-Page Compact Flow Summary

For very fast exam or lab use, readers can remember the six shortest paths like this:

NO BOOT:

```
Power? -> GRUB? -> Linux loads? -> Live USB -> Back up files ->
Boot-Repair
```

NO WI-FI:

```
Wi-Fi on? -> Adapter seen? -> SSIDs visible? -> IP assigned? -> Ping
router? -> Ping 8.8.8.8? -> Ping hostname?
```

NO SOUND:

```
Correct output? -> Volume/mute? -> Test Sound? -> aplay test? ->
alsamixer? -> reset PulseAudio?
```

APP WON'T INSTALL:

```
apt update? -> correct package name? -> dependencies broken? -> apt
--fix-broken? -> bad repo/PPA?
```

BLACK SCREEN AFTER LOGIN:

```
Recent driver/update/extension change? -> older kernel/recovery? ->
inspect GPU driver/PRIME? -> disable extensions? -> rescue if needed
```

PRINTER NOT DETECTED:

```
Powered/connected? -> lsusb/network reachable? -> cups running? ->
printer settings visible? -> lpstat queue? -> test page?
```

Appendix D — Desktop Migration Guide

This appendix is designed for readers arriving from **Windows** or **macOS** who want to understand Ubuntu Desktop without feeling lost. The uploaded Ubuntu Academy outline explicitly includes **transitioning from other OS (Windows/macOS migration tips)** as part of everyday desktop best practices, which makes this appendix a direct match for the book's intended scope.

The attached references support this very well. *Ubuntu: The Complete Guide* introduces Ubuntu by comparing it directly with **Windows** and **macOS**, explains that Ubuntu is a Linux distribution with an easy-to-approach philosophy, and describes familiar desktop elements such as the **Launcher**, **desktop search**, **Files**, and **Settings**. *Mastering Ubuntu* adds that Ubuntu Desktop supports the usual categories of software people expect on Windows, such as browsers, office tools, media players, and email applications, and emphasizes that Ubuntu is suitable for everyday tasks. *Ubuntu Linux Bible* also explains that Ubuntu uses **GNOME** by default but can support other desktop environments when users want a different style or lighter workflow.

A useful illustration is to think of migration not as moving to another planet, but as moving to another city. The street names are different, the layout has its own logic, and some familiar habits must be adjusted—but the core activities remain the same: writing, browsing, sharing, organizing, creating, and communicating.

1. Windows to Ubuntu Desktop Workflow Mapping

Many Windows users arrive on Ubuntu with one key question:

> **"What is the Ubuntu version of what I already know?"**

That is the right question to ask at first. The goal is not to pretend Ubuntu is Windows, but to build a reliable map between familiar workflows and Ubuntu's own way of working.

The attached references strongly support this comparison approach. *Mastering Ubuntu* explicitly states that Ubuntu Desktop supports the usual software categories familiar to Windows users, such as **Firefox**, **Chrome**, **VLC**, **LibreOffice**, and **Thunderbird**, and also highlights Ubuntu's strong search, customization, and everyday productivity features. *Ubuntu: The Complete Guide* describes Ubuntu's **Launcher**, **Settings**, **Files**, and **desktop search** in ways that make direct workflow mapping possible.

Windows concept	Ubuntu equivalent	Notes
Start menu	**Activities / Show Applications**	Open apps by clicking **Activities** or pressing **Super** and typing the app name.
Taskbar	**Top bar + Dock / Launcher**	Ubuntu uses a top bar and a left-side Dock/Launcher rather than one bottom taskbar by default.
File Explorer	**Files (Nautilus)**	Ubuntu uses **Files** as the main graphical file manager.
Control Panel / Settings	**Settings**	Most system configuration lives in one main Settings app.
Desktop search	**Activities search**	Type immediately after opening Activities to search apps, files, and more.
Microsoft Edge / Chrome	**Firefox / Chromium / Chrome**	Ubuntu supports mainstream browsers.
Microsoft Office	**LibreOffice**	Ubuntu's standard open office suite is LibreOffice.
Outlook	**Thunderbird**	Ubuntu references explicitly note Thunderbird as built-in or standard email software.
Windows Media Player	**VLC / Videos / Rhythmbox**	Ubuntu commonly separates music and video apps more than Windows does.

Device Manager	**Settings + lspci / lsusb / lshw / inxi**	Ubuntu often combines GUI settings with strong CLI hardware tools.
Windows Update	**Software Updater / APT / unattended-upgrades**	Ubuntu balances GUI and CLI update models.
Command Prompt / PowerShell	**Terminal (Bash shell)**	Ubuntu provides Terminal access to the Bash shell for advanced tasks.

What feels different for Windows users?

The references suggest several common adjustment points:

- Ubuntu uses a **top bar + Dock** rather than the classic Windows taskbar.
- Ubuntu's **search-first workflow** is more central than menu navigation alone.
- The Linux filesystem is broader and more complex than the average Windows folder view, so new users are encouraged to focus mostly on their **Home** folder at first. *Ubuntu: The Complete Guide* says this explicitly.
- Software is usually installed through **repositories/App Center** rather than downloaded individually from random websites.

A useful illustration is to think of Windows as a familiar office building with labeled drawers and control panels. Ubuntu has the same core rooms, but the signs and organization are arranged around GNOME's ideas of search, workspaces, and a left-side launcher.

Practical Windows-to-Ubuntu migration advice

- Learn **Super** as your best friend.
- Use **Files** instead of trying to find a literal "Explorer clone."
- Prefer the **App Center / Ubuntu Software** before hunting random `.deb` files.
- Keep daily work inside **Home**, **Documents**, **Downloads**, and project folders at first.
- Don't judge Ubuntu by how closely it imitates Windows. Judge it by how quickly you can complete your real tasks.

2. macOS to Ubuntu Desktop Workflow Mapping

Users coming from macOS often adjust differently than Windows users. They are usually more comfortable with a **top bar**, system-wide search, and a Dock-like launcher model. In that sense, the transition to Ubuntu GNOME can feel more natural in some areas, even though the design language is still clearly different.

The attached references support this comparison indirectly but usefully. *Mastering Ubuntu* notes that Ubuntu is unlike **Windows and macOS**, though it has some elements in common, and then describes the **top bar**, **Dock**, and **Activities Overview**. It also specifically mentions that on a Mac keyboard, the **Command key** corresponds to the **Super**-style usage in some shortcut explanations. *Ubuntu: The Complete Guide* also explains Ubuntu in direct comparison with Windows and macOS as operating systems users likely already understand.

macOS to Ubuntu quick map

macOS concept	Ubuntu equivalent	Notes
Finder	**Files (Nautilus)**	Main graphical file manager.
Spotlight	**Activities search**	Search is a core Ubuntu GNOME habit.
Dock	**Dock / Launcher**	Ubuntu keeps the Dock vertically by default on the left.
System Settings	**Settings**	Central GUI control point.
Mission Control / Spaces	**Activities Overview + Workspaces**	Ubuntu GNOME has built-in workspaces and overview-based window management.
Launchpad	**Show Applications**	App grid available from GNOME’s launcher model.
Terminal	**Terminal**	Ubuntu uses Bash by default for CLI work.

Safari	**Firefox / Chrome / Chromium**	Ubuntu commonly uses Firefox by default or near-default.
Pages / Numbers / Keynote	**LibreOffice Writer / Calc / Impress**	Open-source office equivalents.
Time Machine (conceptually)	**Déjà Dup / backups + snapshots**	The Ubuntu Academy outline includes Déjà Dup and Timeshift-style backup/snapshot concepts.

What feels familiar to macOS users?

Several things may feel more natural to a macOS user than to a Windows user:

- a **top bar** is already a familiar concept.
- a **Dock/Launcher** is already a familiar concept.
- **search-first launching** is closer in spirit to Spotlight than to old-style menu drilling.
- **workspaces** and overview-based desktop organization may feel closer to macOS Spaces and Mission Control logic than to older Windows habits.

What feels different for macOS users?

- Ubuntu GNOME's visual design is not macOS, even if some layout ideas overlap.
- Package installation is still more repository-centered than typical macOS app-download culture.
- The filesystem and app structure are more Linux-like than macOS bundle-based habits.
- Ubuntu gives much more explicit exposure to open-source components, package formats, and desktop environment variations.

A useful illustration is to think of macOS-to-Ubuntu migration like moving from one design-oriented campus to another. The buildings are still clean and organized, and there is still a top bar and a Dock-like launcher—but the rooms are arranged around GNOME's workflows rather than Apple's.

Practical macOS-to-Ubuntu migration advice

- Treat **Activities search** as your new Spotlight.
- Treat **Files** as your new Finder, but keep in mind Linux folders and paths have their own logic.
- Use **workspaces** intentionally; they are one of GNOME's strengths.

- Learn Ubuntu's package-management model early so software installation feels normal, not "different for no reason."
- Do not expect a literal clone of macOS; expect a desktop that shares some workflow patterns while remaining fully Linux.

Screenshot notes:

`[Screenshot-APPD-03: Side-by-side comparison: macOS desktop vs Ubuntu GNOME desktop with labels for Spotlight/Activities search, Dock/Launcher, Finder/Files, System Settings/Settings.]`
`[Screenshot-APPD-04: GNOME Activities Overview compared conceptually to Mission Control / Spaces.]`

3. Common Terminology Translation

One of the hardest parts of migration is not the operating system itself. It is the **language**. A user may already know how to do the work but still feel lost because the terms are unfamiliar. That is why the Ubuntu Academy outline explicitly includes **common terminology translation** under migration guidance.

Below is a practical translation map.

General desktop terms

Familiar term	Ubuntu / Linux term	Notes
Program / App	**Application / Package**	In Linux, “package” often refers to the installable unit.
Install file	**Package / .deb / Snap / Flatpak**	Ubuntu supports multiple package systems.
Taskbar	**Top bar + Dock / Launcher**	Ubuntu GNOME splits these roles.
Start menu / Launchpad / Spotlight	**Activities / Show Applications / Activities search**	Search is central.
File Explorer / Finder	**Files / Nautilus**	Default GNOME file manager.
Control Panel / System Settings	**Settings**	Main GUI settings application.
Desktop spaces / virtual desktops	**Workspaces**	Built into GNOME workflow.
App store	**Ubuntu Software / App Center / Software Center**	Varies slightly by version and branding.
Command Prompt / Terminal.app	**Terminal**	Access to Bash shell on Ubuntu.
Driver manager	**Additional Drivers / driver tools / hardware detection commands**	Often combined GUI + CLI workflow.

Filesystem and location terms

Windows / macOS habit	Ubuntu term	Notes
C:\Users\Name\Documents	**/home/name/Documents**	Linux paths use / not \.
This PC / Computer	**filesystem tree + Home focus**	Beginners should mostly stay in Home first.
Drive letter	**mount point**	Linux usually mounts devices under directories rather than assigning letters.
Install disk / mounted DMG / removable drive	**mounted filesystem under /media/... or other mount point**	Consistent with Ubuntu Academy external-device focus.

Open-source ecosystem terms

Term	Meaning in Ubuntu context	Notes
Distribution / distro	**A full Linux operating system built around Linux kernel + packages**	Ubuntu: The Complete Guide explicitly says Ubuntu is a distro.
Flavor	**Official Ubuntu variant with a different desktop/environment focus**	Ubuntu Linux Bible explicitly mentions flavors such as Kubuntu, Lubuntu, and Ubuntu MATE.
Desktop environment	**The visual desktop shell and core apps, such as GNOME, KDE, Xfce**	Supported by both Mastering Ubuntu and Ubuntu Linux Bible.
Repository / repo	**Trusted software source used by package managers**	Central to Ubuntu software management.
Shell	**Command-line environment, typically Bash**	Ubuntu Linux Bible explicitly introduces Bash shell.

A useful illustration is to think of terminology translation as learning the campus vocabulary. The rooms still exist—but if you learn the local names, finding them becomes much easier.

4. Practical Migration Strategy

The best migration strategy is not “forget everything you know.” It is “map what you know onto Ubuntu’s structure.” The uploaded Ubuntu Academy outline explicitly supports migration tips as part of everyday best practices, and the attached books repeatedly present Ubuntu as approachable, customizable, and suitable for everyday work once its desktop logic is understood.

A practical migration plan looks like this:

- first learn the **Ubuntu equivalents** of your most-used tools
- keep daily work inside familiar places such as **Home**, **Documents**, **Downloads**, and project folders
- use **Activities search** constantly until it feels natural
- do not customize too much on day one
- learn the App Center / Ubuntu Software early
- keep one small note file of “Windows/macOS term → Ubuntu term”
- adopt workspaces and shortcuts gradually rather than all at once

A good final illustration is to think of migration as building a bilingual dictionary. At first, you translate everything. After a while, you stop translating and simply start thinking in Ubuntu’s own workflow.

CLI example: show the current home path for orientation

```
echo $HOME
pwd
```

Example output:

```
/home/student
/home/student
```

CLI example: launch Files and Settings from the command line

```
nautilus &
gnome-control-center &
```

Example output:

```
[1] 8122
[2] 8123
```

This is a nice reminder that Ubuntu’s GUI and CLI worlds are not enemies. They are partners, and that is one of Ubuntu Desktop’s strengths.

Screenshot notes:

[Screenshot-APPD-07: Migration checklist page with three columns: Windows habit, macOS habit, Ubuntu workflow.]
[Screenshot-APPD-08: Clean Ubuntu desktop with Files, Settings, and Activities search highlighted as the three key migration anchors.]

Appendix E — Glossary

Term	Brief definition
Activities Overview	The main GNOME navigation hub for viewing open applications, switching workspaces, searching for files and apps, and opening the App Grid.
APT	Advanced Package Tool, Ubuntu's `.deb`-based package management system for updating, installing, removing, and upgrading software through repositories.
App Grid	The GNOME view that shows all installed applications in a grid layout.
Bluetooth	A short-range connection technology for pairing devices such as headsets, speakers, keyboards, mice, or for light file transfer.
Bootable USB	A USB flash drive written with an Ubuntu installer image so a computer can boot directly into live mode or the installer.
CLI	Command-Line Interface, a way to interact with the system through text commands in Terminal. It contrasts with the click-and-window style of a GUI.
Desktop Environment	The graphical layer of Linux that provides windows, menus, panels, icons, a file manager, settings, and the overall desktop experience.
Display Server	The component that connects graphical applications to the display, mouse, and keyboard. On modern Ubuntu this is usually Wayland, with X.Org available as a compatibility option.
Distribution / Distro	A complete Linux operating system package that includes the kernel, tools, package manager, desktop, repositories, and ready-to-use configuration. Ubuntu is one Linux distribution.
Dock	The application launcher panel on the side of the Ubuntu desktop that also shows running applications.
File Manager	A graphical application for browsing folders, moving files, accessing external storage, and opening network shares. In GNOME, this is Files/Nautilus.
Files (Nautilus)	The default GNOME/Ubuntu Desktop file manager for visually managing user files and folders.
Flatpak	A cross-distribution Linux application distribution system that runs apps in isolated package format. It is commonly used with sources such as Flathub.

Flathub	A popular repository for Flatpak applications.
GNOME	Ubuntu Desktop's default desktop environment, known for its modern, clean interface and its focus on productivity, Activities, search, and workspaces.
GNU/Linux	A term emphasizing that a Linux system is not only the kernel, but also user tools and other components, many of which come from the GNU project.
GUI	Graphical User Interface, a visual interface with windows, icons, menus, buttons, and graphical settings.
Home Directory	The user's personal folder, usually located at `/home/username`, where Documents, Downloads, Pictures, and hidden configuration files are stored.
Kernel	The core of the Linux operating system that manages communication with hardware and system resources.
LTS (Long Term Support)	An Ubuntu release focused on long-term stability, suitable for learning, institutions, offices, and daily workstation use.
Live Mode	A mode that runs Ubuntu directly from a USB drive without permanent installation, useful for testing hardware and trying the desktop first.
NetworkManager	The network service/manager commonly used in Ubuntu Desktop for Wi-Fi, Ethernet, and everyday user connections.
Package Manager	A system for finding, installing, updating, and removing software, such as APT, Snap, and Flatpak.
Permission	Access rights to files, folders, or system actions. Usually related to the user, group, ownership, and access mode.
Quick Settings	The quick-access panel in the top-right area of GNOME for controlling Wi-Fi, volume, brightness, power, and session controls.
Repository	A software package source used by the system to retrieve applications and updates.
Session	The user's active desktop login environment, including running applications, the desktop type, and the display system in use.
Snap	An application package format managed by Canonical, isolated from the core system and easy to install and update.
Super Key	The keyboard key used to open Activities Overview and speed up GNOME navigation. It is often the same as the Windows logo key on PC keyboards.

Terminal	The application used to run CLI commands in Ubuntu. It is very important for administration, troubleshooting, and automation.
Toolkit (GTK / Qt)	A set of interface libraries used by applications to create buttons, menus, dialog boxes, and other visual elements. GTK is common in GNOME, while Qt is common in KDE.
Ubuntu Desktop	The Ubuntu edition intended for interactive human use with a GUI, desktop applications, multimedia, browsing, office work, and daily productivity.
Ubuntu Server	The Ubuntu edition more focused on server services, command-line administration, stability, and remote management than on a full desktop GUI.
Wayland	The modern display protocol/system that is the default on Ubuntu Desktop, designed to be cleaner and more secure for modern desktop needs.
Window Manager	The component that controls the position, size, arrangement, minimization, maximization, and behavior of application windows.
Workspace	A virtual desktop used to separate tasks into multiple work areas so the screen stays less crowded and focus is maintained.
X.Org / X11	The older display system still available in Ubuntu as a compatibility option for certain hardware or applications.

References

Bin Uzayr, S. (2023). *Mastering Ubuntu: A beginner's guide*. CRC Press.

Clinton, D., & Negus, C. (2026). *Ubuntu Linux Bible* (11th ed.). John Wiley & Sons.

Future Publishing Limited. (2021). *Ubuntu: The complete guide* (11th ed.). Future Publishing Limited.

Author



Onno W. Purbo currently serves as Rector at Institut Teknologi Tangerang Selatan (ITTS). He earned his Ph.D. in Electrical Engineering from the University of Waterloo, Canada. He is a copyleft advocate, educator, and ICT evangelist. He has published more than 50 books, including free ICT e-books for schools in 2008. Some of his latest books include *"Internet-TCP/IP: Konsep Dan Implementasi"* (2018); *"Sistem Operasi, Konsep Dan Membuat Linux OpenWRT Dan ROM Android"* (2019); *"IPv6 Untuk Mendukung Operasi Jaringan Dan Domain Name System"* (2019); *"Kubernetes untuk Pemula"* (2024); and *"Membuat Operator Seluler 5G Sendiri"* (2024).

He led the first Internet connection at Institut Teknologi Bandung from 1993–2000 and used it to establish Indonesia's first educational Internet network. He played a key role in liberating WiFi frequencies, introducing RT/RW-net, the Wajanbolic antenna, as well as OpenBTS-based cellular networks and private 5G networks. He also led the first telephone network over the Internet, VoIP Merdeka, which later became known as VoIP Rakyat based on SIP and using the area codes +62520 and +62521.

He is currently active in promoting e-Learning and operates an e-Learning server at http://lms.onnocenter.or.id/moodle/ serving more than 66,000 students.