

Linux - Administration

m2iformation.fr



Program

- Installation and Configuration
 - Overview of server operating systems.
 - Preparing for installation and best practices.
 - Installation process and troubleshooting.
 - Post-installation tasks for system stability.
 - Basic system configuration.
- Identity Management
 - Managing users and groups.
 - Profiles and environments.
 - Hardening identity security.
- Package Management
 - Introduction to package management systems.
 - Installing, updating, and resolving package conflicts.
 - Managing RPM, DEB, AppImage, Flatpak, and Snap packages.
 - Installing applications from tarball archives.
 - Compiling and installing from source.
- Disk and File System Management
 - Overview of file system types (ext2, ext3, ext4).
 - Partitioning and logical volume management.
 - Formatting and mounting file systems.
 - Configuring quotas for users and groups.
 - Introduction to RAID.

Program

- Daemon and Service Management
 - Understanding the boot process and GRUB2 configuration.
 - Managing services and targets with systemd.
 - Managing SysVinit services.
 - Troubleshooting in rescue and emergency modes.
- Process and System Monitoring
 - Defining processes, threads, and scheduling.
 - Monitoring system activity (top, pstree, ps).
 - Managing signals (kill, pkill, killall).
 - Background and foreground tasks (bg, fg, jobs).
 - Scheduling tasks with cron and at.
- Networking
 - Configuring basic network settings.
 - Deploying and managing DNS, DHCP, SSH, and Apache services.
 - Analyzing and monitoring network traffic.
- Backup and Restoration
 - Compressing and decompressing files (gzip, bzip2, lzma, lzw).
 - Managing archives using tar.
 - Synchronizing files with backup tools.



Installation and Configuration



Overview of Server Operating Systems

OVERVIEW OF SERVER OPERATING SYSTEMS



What Makes a Server OS Different?

- Stability and uptime
- Security
- Resource efficiency
- Remote management

Common Server Operating Systems



- Linux Distributions
 - Ubuntu Server
 - Debian
 - CentOS / Rocky Linux / AlmaLinux
 - Red Hat Enterprise Linux (RHEL)
 - SUSE Linux Enterprise Server (SLES)

Key Considerations When Choosing a Server OS

- Hardware compatibility
- Cost
- Available community or vendor support

Common Server Operating Systems

Linux Distributions (Distros)

Linux is a top choice for servers due to its **open-source nature, stability, performance, and flexibility**.

- **Ubuntu Server**: Easy to use, popular in cloud environments, backed by Canonical.
- **Debian**: Known for stability, ideal for long-term server deployments.
- **CentOS / Rocky Linux / AlmaLinux**: Popular for enterprise environments (Red Hat compatible).
- **Red Hat Enterprise Linux (RHEL)**: Commercial support, widely used in critical systems.
- **SUSE Linux Enterprise Server (SLES)**: Used in enterprises, especially in Europe.

Unix Variants

- **FreeBSD**: Known for performance and advanced networking capabilities.
- **OpenBSD**: Focuses on security and code correctness.

Key Considerations When Choosing a Server OS

When choosing a server OS, administrators evaluate:

- **Hardware compatibility**
- **Available community or vendor support**
- **Security requirements**
- **Type of services to host (e.g., web, database, file server)**
- **Cost** (license fees vs. open-source)

🔧 Preparing for Installation and Best Practices

Before installing a Linux server, proper **preparation is essential** to ensure a smooth and secure deployment. This phase helps prevent misconfigurations, hardware compatibility issues, or downtime later in production.

◆ 1. Define the Purpose of the Server

Clearly identify what the server will be used for:

- Web server (Apache, Nginx)
- File server (NFS, Samba)
- Database server (MySQL, PostgreSQL)
- DNS, DHCP, Email, VPN, etc.

Why?

➡ Because each use case requires different services, configurations, and sometimes different distributions.

🔧 Preparing for Installation and Best Practices

◆ 2. Choose the Right Linux Distribution

Select a Linux version suited for your goals:

- **Ubuntu Server**: Cloud-friendly, easy to use.
- **Debian**: Stable and reliable.
- **CentOS Stream / Rocky Linux / AlmaLinux**: Enterprise environments.
- **RHEL / SLES**: If you need vendor support.

🔧 Preparing for Installation and Best Practices

◆ 3. Verify Hardware Requirements

Check:

- CPU architecture (x86_64, ARM)
- RAM (minimum and recommended for your services)
- Disk space
- Network interfaces
- Compatibility (especially for RAID controllers, Wi-Fi cards, GPUs if needed)

Use tools like:

```
lscpu  
lsblk  
lspci  
lsusb
```

🔧 Preparing for Installation and Best Practices

◆ 4. Plan Your Disk Layout

Decide:

- Separate partitions for `/`, `/home`, `/var`, `/tmp`, `/boot`
- Use of LVM (Logical Volume Manager) for flexibility
- File systems: ext4, xfs, btrfs

Why?

- ➡ It helps in isolating logs, backups, user data, and makes maintenance easier.

🔧 Preparing for Installation and Best Practices

◆ 5. Prepare a Backup Plan (if reusing hardware)

- If you're installing on an existing server, make sure to backup all critical data.
- Test the backup restore procedure!

◆ 6. Plan for Network Configuration

- Static IP or DHCP?
- DNS servers and hostname?
- Firewall ports to open?
- Will the server be accessible from outside?

Prepare your IP settings **before installation.**

🔧 Preparing for Installation and Best Practices

◆ 7. Choose Installation Method

Depending on the scenario:

- **CD/DVD or USB bootable media**
- **PXE (Preboot Execution Environment)** for mass installations
- **Cloud-init** or preseed/kickstart files for automation

◆ 8. Follow Best Practices

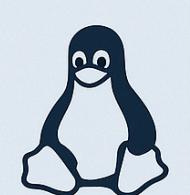
- ✓ Use only the packages you need (minimal install = fewer attack surfaces)
- ✓ Use a strong root password or better, disable root login and use `sudo`
- ✓ Create an unprivileged user during setup
- ✓ Set up SSH keys instead of passwords
- ✓ Update the system after installation (`apt update && apt upgrade` or `dnf update`)
- ✓ Document the configuration and installation steps

🔧 Preparing for Installation and Best Practices

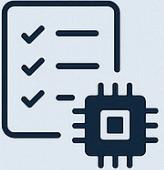
Preparing for Installation and Best Practices



Define the Purpose
of the Server



Choose the Right
Linux Distribution



Verify Hardware
Requirements



Plan Your
Disk Layout



Prepare a Backup
Plan



Plan for Network
Configuration



Choose Installation
Method



Follow
Best Practices



Installation Process and Troubleshooting

◆ 1. Starting the Installation

Once preparation is done, the installation process begins by:

- **Booting from the installation media (USB/DVD/ISO)**
 - Selecting the **language, keyboard layout, and time zone**
 - Choosing between **manual or guided partitioning**
 - Creating users and setting the root password
 - Selecting packages or performing a minimal installation
-  Most distributions (Ubuntu, Debian, RHEL, etc.) offer both graphical and text-based installation modes.



Installation Process and Troubleshooting

◆ 2. Common Installation Steps

1. Partition Disks

- Use LVM for flexibility
- Format with `ext4`, `xfs`, etc.

2. Configure Bootloader

- GRUB is usually installed on the main disk (e.g., `/dev/sda`)

3. Set Up Network

- DHCP or static IP configuration
- Set hostname and DNS

4. Install Packages

- Select predefined sets (server, desktop, minimal)
- Or choose manually



Installation Process and Troubleshooting

5. Install Updates

- After installation, run:

```
sudo apt update && sudo apt upgrade
```

or

```
sudo dnf update
```



Installation Process and Troubleshooting

◆ 3. Post-Installation Checks

- Can the system boot without error?
- Can you log in via the console or SSH?
- Is the network working?
- Are all partitions mounted correctly?
- Are essential services running?

Use:

```
systemctl status  
ip a  
df -h
```



Installation Process and Troubleshooting



4. Troubleshooting Installation Issues

Problem	Possible Cause	Solution
 Kernel panic	Hardware unsupported or misconfigured bootloader	Try different kernel or boot options
 No network	Missing drivers, bad config	Check <code>ip a</code> , <code>dmesg</code> , network config files
 GRUB not found	Bootloader not installed	Boot from live USB and reinstall GRUB
 Disk not detected	RAID/SATA/NVMe drivers missing	Load appropriate drivers or switch to AHCI in BIOS
 Can't login	Wrong user/pass or SSH key issue	Boot into recovery mode and reset password



Installation Process and Troubleshooting

◆ 1. Common Boot Issues

◆ GRUB Bootloader Errors

Symptoms:

- Blank screen or “grub rescue>” prompt after reboot
- “No such partition” or “unknown filesystem”

Causes:

- GRUB was not installed properly
- Wrong boot device selected
- Partition table or boot sector damaged

Installation Process and Troubleshooting

Solutions:

```
# From a Live CD:  
sudo mount /dev/sdXn /mnt          # Mount root partition  
sudo mount --bind /dev /mnt/dev  
sudo mount --bind /proc /mnt/proc  
sudo mount --bind /sys /mnt/sys  
sudo chroot /mnt  
grub-install /dev/sdX  
update-grub
```



Installation Process and Troubleshooting

◆ 2. Network Issues During Installation

◆ No network connectivity

Symptoms:

- DHCP fails, no IP assigned
- Can't update or fetch packages

Causes:

- No drivers for network card
- DHCP server not responding
- Wrong network configuration

Installation Process and Troubleshooting

Solutions:

```
# Check network interfaces:  
ip a  
# Enable interface manually  
sudo ip link set dev eth0 up  
# Try restarting DHCP  
sudo dhclient eth0
```

“ For static IP: manually configure `/etc/network/interfaces` (Debian) or `nmcli` (RedHat-based systems) ”



Installation Process and Troubleshooting

◆ 3. Disk and Partitioning Issues

◆ Disk not detected

Symptoms:

- No disk shown in partitioning screen

Causes:

- Missing SATA/RAID/NVMe drivers
- BIOS misconfiguration

Solutions:

- In BIOS/UEFI, set disk mode to **AHCI**
- Use `lsblk`, `fdisk -l`, `dmesg | grep sd` to check device presence
- If using LVM, activate it:

```
vgscan  
vgchange -ay
```



Installation Process and Troubleshooting

◆ Filesystem or partition errors

Symptoms:

- Installation stops at formatting step
- Mount errors during boot

Solutions:

```
# From a live system:  
fsck /dev/sdXn
```



Installation Process and Troubleshooting

◆ 4. Installation Freeze or Crash

Symptoms:

- Graphical installer hangs
- Black screen with no logs

Solutions:

- Try **text-based installer** or **alternate ISO**
- Boot with kernel parameters like:
 - `nomodeset` (for graphics)
 - `acpi=off` (for power issues)
 - `noapic` or `nolapic` (for CPU issues)

Edit the GRUB entry by pressing `e` and appending to the `linux` line.



Installation Process and Troubleshooting

◆ 5. Password or User Login Issues

Symptoms:

- Can't log in after reboot
- SSH keys not accepted

Solutions:

- Boot into **recovery mode** or use Live CD:

```
mount /dev/sda1 /mnt
chroot /mnt
passwd yourusername
```



Installation Process and Troubleshooting

◆ 6. Package Installation Failure

Symptoms:

- "Package X could not be installed"
- "Failed to fetch"

Solutions:

- Check internet access
- Verify `/etc/apt/sources.list` or repository configuration
- Clear and retry:

```
sudo apt clean  
sudo apt update
```



Installation Process and Troubleshooting



7. Useful Log Files

- `/var/log/syslog` – system-wide messages
- `/var/log/installer/` – specific to the installation process
- `dmesg` – kernel messages (drivers, devices)
- `journalctl` – on systems using `systemd`

Post-installation Tasks for System Stability

Once Linux is installed, your system is technically operational – but **not production-ready**. There are several essential tasks to ensure **stability, performance, and security** over time.

◆ 1. Update the System Immediately

After installation, the packages may be outdated.

Why?

New vulnerabilities may have been patched or critical updates released.

Commands:

```
# For Debian/Ubuntu  
sudo apt update && sudo apt upgrade
```

```
# For RHEL-based systems  
sudo dnf update
```

✓ Post-installation Tasks for System Stability

◆ 2. Enable and Configure a Firewall

Ensure that only the necessary ports are open.

Tools:

- `ufw` (simple and user-friendly)
- `firewalld` (CentOS, RHEL)
- `iptables` (low-level control)

Example:

```
# For UFW
sudo ufw enable
sudo ufw allow ssh
sudo ufw allow http
```

✓ Post-installation Tasks for System Stability

◆ 3. Disable Unnecessary Services

Each running service increases the **attack surface** and uses system resources.

List services:

```
systemctl list-units --type=service
```

Disable unnecessary ones:

```
sudo systemctl disable bluetooth.service
```

✓ Post-installation Tasks for System Stability

◆ 4. Set Up Automatic Security Updates

To keep your system safe over time, configure unattended upgrades.

Ubuntu/Debian:

```
sudo apt install unattended-upgrades  
sudo dpkg-reconfigure unattended-upgrades
```

RHEL-based (with dnf-automatic):

```
sudo dnf install dnf-automatic  
sudo systemctl enable --now dnf-automatic.timer
```

✓ Post-installation Tasks for System Stability

◆ 5. Create Regular Users and Disable Root SSH Login

Using root is dangerous. Instead, use a non-root user with `sudo`.

```
sudo adduser adminuser  
sudo usermod -aG sudo adminuser
```

Then edit `/etc/ssh/sshd_config`:

```
PermitRootLogin no
```

Restart SSH:

```
sudo systemctl restart ssh
```

✓ Post-installation Tasks for System Stability

◆ 6. Configure Time Synchronization (NTP)

Correct time is crucial for logs, authentication, cron jobs, etc.

Recommended service:

```
sudo timedatectl set-ntp true
```

To verify:

```
timedatectl status
```

✓ Post-installation Tasks for System Stability

◆ 7. Set Up Monitoring and Logging

Install tools to monitor:

- CPU, memory, disk (e.g., `htop`, `iotop`)
- Services (`systemctl`, `journalctl`)
- Remote logging (e.g., `rsyslog`, `syslog-ng`, or forwarding to Splunk/Grafana)

✓ Post-installation Tasks for System Stability

◆ 8. Schedule Backups

Plan for failure. Even stable systems can crash.

- Backup `/etc`, user data, databases
- Use tools like `rsync`, `borg`, or `tar`
- Automate with `cron` or `systemd` timers

✓ Post-installation Tasks for System Stability

◆ 9. Enable Swap (if needed)

Some systems don't create swap by default.

Check:

```
swapon --show
```

If no swap:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

Post-installation Tasks for System Stability

◆ 10. Harden the System

- Use tools like **Lynis** to audit the system:

```
sudo apt install lynis
sudo lynis audit system
```

- Disable unnecessary kernel modules
- Configure SELinux (RHEL) or AppArmor (Ubuntu)

⚙ Basic System Configuration

After installing the system, some initial configuration is required to ensure the server runs reliably, follows organizational standards, and is ready for further deployment.

◆ 1. Set the Hostname

The hostname is the unique name assigned to your machine within the network or your infrastructure. It is used in logs, SSH sessions, and prompt appearance (`user@hostname`).

To view the current hostname:

```
hostnamectl
```

To change it:

```
sudo hostnamectl set-hostname server01
```

“ Optionally, update `/etc/hosts` to reflect the new name: ”

```
127.0.0.1    localhost
127.0.1.1    server01
```

◆ 2. Configure the System Time and Timezone

Accurate system time is crucial for:

- Log file timestamps
- Scheduled tasks (cron jobs)
- Authentication and security mechanisms
- Synchronizing distributed systems

Check current settings:

```
timedatectl
```

Set your timezone (example for New York):

```
sudo timedatectl set-timezone America/New_York
```

“ To list all available time zones: ”

```
timedatectl list-timezones
```

Enable time sync with NTP:

```
sudo timedatectl set-ntp true
```

◆ 3. Configure the System Locale and Keyboard Layout

Setting the correct locale ensures that your system:

- Displays characters correctly
- Sorts text based on your language
- Uses the right number/date formats

View current locale:

```
locale
```

Reconfigure (Debian-based):

```
sudo dpkg-reconfigure locales
```

Generate missing locales:

```
sudo locale-gen en_US.UTF-8
```

Keyboard configuration (Debian):

```
sudo dpkg-reconfigure keyboard-configuration
```

◆ 4. Configure the Bootloader (GRUB)

It's important to ensure that your bootloader is correctly installed and configured.

To view GRUB config (Debian/Ubuntu):

```
cat /boot/grub/grub.cfg
```

To update GRUB:

```
sudo update-grub
```

To reinstall GRUB (if needed):

```
sudo grub-install /dev/sda  
sudo update-grub
```

You can also customize:

- Default kernel
- Timeout
- Splash screen
- Kernel boot parameters

Edit `/etc/default/grub` and then run `update-grub`.

◆ 5. Enable and Configure Logging

System logs help you monitor system behavior and diagnose issues.

Systemd-based systems use `journald`:

- View logs:

```
journalctl
```

- Logs persist across reboots:

```
sudo mkdir -p /var/log/journal  
sudo systemd-tmpfiles --create --prefix /var/log/journal  
sudo systemctl restart systemd-journald
```

Also check log rotation:

- Managed by `logrotate`
- Configured in `/etc/logrotate.conf` and `/etc/logrotate.d/`

Test log rotation:

```
sudo logrotate --debug /etc/logrotate.conf
```

◆ 7. Review Kernel Modules

Only necessary kernel modules should be loaded.

List loaded modules:

```
lsmod
```

To remove one (example: bluetooth):

```
sudo modprobe -r bluetooth
```

You can blacklist modules you don't want at boot by editing `/etc/modprobe.d/blacklist.conf`.

◆ 8. Configure System Services

Ensure critical services are **enabled**, and unnecessary ones are **disabled**.

View running services:

```
systemctl list-units --type=service
```

Enable a service (example: cron):

```
sudo systemctl enable --now cron
```

Disable a service (example: cups if unused):

```
sudo systemctl disable --now cups
```

◆ 9. Set Up Log Monitoring Alerts (Optional for long-term stability)

Advanced setups include installing lightweight log monitoring tools like:

- `logwatch`
- `logcheck`

These can email you daily summaries of system events, helping to catch early signs of problems.



Identity Management

◆ 1. User and Group Management

Types of Users

- **Root (UID 0)**: full control of the system.
- **System users (UID < 1000)**: created by services like `mysql`, `www-data`.
- **Regular users (UID ≥ 1000)**: human users.

Key Files

File	Description
<code>/etc/passwd</code>	Basic user info (username, UID, shell, etc.)
<code>/etc/shadow</code>	Encrypted passwords + password policies
<code>/etc/group</code>	List of groups and members
<code>/etc/gshadow</code>	Secure group passwords

◆ 1. User and Group Management

🔧 User Commands

Task	Command
Add user (with home)	<code>adduser alice</code>
Add user (manual)	<code>useradd -m alice</code>
Delete user	<code>deluser alice</code> or <code>userdel alice</code>
Delete user and home	<code>userdel -r alice</code>
Modify user	<code>usermod</code> (e.g. <code>usermod -l newname oldname</code>)
Set password	<code>passwd alice</code>
View user info	<code>id alice</code> or <code>getent passwd alice</code>
List all users	<code>cut -d: -f1 /etc/passwd</code>

◆ 1. User and Group Management

🔧 User Commands

Task	Command
Force password reset	chage -d 0 alice
Expire user	usermod --expiredate 1 alice
Lock user	usermod -L alice
Unlock user	usermod -U alice
Set shell	usermod -s /bin/bash alice

Group Commands

Task	Command
Add group	<code>groupadd devs</code>
Delete group	<code>groupdel devs</code>
Rename group	<code>groupmod -n devops devs</code>
Add user to group	<code>usermod -aG devs alice</code>
Remove user from group	<code>gpasswd -d alice devs</code>
View user groups	<code>groups alice</code>
List all groups	<code>cut -d: -f1 /etc/group</code>
View group members	<code>getent group devs</code>

◆ 2. Profiles and Shell Environments

A **profile** defines a user's environment on login: shell, environment variables, aliases, etc.

User Shell Files

File	Purpose
<code>~/.bashrc</code>	Run for interactive shells
<code>~/.profile</code>	Run for login shells
<code>~/.bash_logout</code>	Run at logout
<code>/etc/profile</code>	System-wide login configuration
<code>/etc/bash.bashrc</code>	System-wide interactive shell

⚙️ Common Customizations

- Define variables:

```
export PATH=$PATH:/opt/tools  
export EDITOR=nano
```

- Add aliases:

```
alias ll='ls -lah'  
alias gs='git status'
```

- Customize prompt (`PS1`), umask, etc.

Reload with:

```
source ~/ .bashrc
```

◆ 3. Hardening Identity Security

🔒 Enforce Password Policies

Install module:

```
sudo apt install libpam-pwquality
```

Edit `/etc/security/pwquality.conf`:

```
minlen = 12
dcredit = -1
ucredit = -1
ocredit = -1
lcredit = -1
```

Password aging:

```
chage -M 90 alice      # Max days
chage -m 7 alice       # Min days
chage -W 10 alice      # Warning before expiry
chage -l alice         # List settings
```

🔒 Account Restrictions

Task	Command
Lock user	passwd -l alice or usermod -L alice
Unlock user	passwd -u alice
Expire now	chage -E 0 alice
Disable shell access	usermod -s /usr/sbin/nologin alice

🔑 Sudo Configuration

Edit via `visudo`:

Allow full access:

```
alice ALL=(ALL) ALL
```

Restrict to command:

```
bob ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart nginx
```

Monitor Login & Auth Activity

- View successful logins:

```
last
```

- View failed logins:

```
faillog -a
```

- Check logs:

```
less /var/log/auth.log  
journalctl -u ssh
```

🧱 Home Directory Permissions

Protect user home:

```
chmod 700 /home/alice  
chown alice:alice /home/alice
```

Use `umask` (e.g. `027`) to restrict default file permissions:

```
# Add to /etc/profile or ~/.profile  
umask 027
```

Identity Management

Identity Management in Linux



Managing Users and Groups

- User types
- User commands
- /etc.passwd
/etc.shadow,
/etc.group



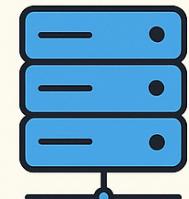
Profiles and Environments

- User shell files
- Customizations
- Reload environment



Hardening Identity Security

- Password policies
- Account restrictions
- Sudo configuration
- Monitor activity
- Home directory permissions



Optional: Centralized Identity

- LDAP, Kerberos, SSSD, etc.



Package Management

◆ 1. Introduction to Package Management Systems

In Linux, a **package** is a bundle that contains:

- A **program** (binaries and libraries)
- **Metadata** (name, version, architecture, description)
- **Dependencies** (other packages required)
- Sometimes **scripts** (pre-install/post-install hooks)

This allows us to **install, update, or remove** software in a clean, reproducible, and automated way.

💡 What is a Package Manager?

A **package manager** is a tool that automates:

- Searching for software
- Installing software
- Upgrading or downgrading packages
- Handling dependencies
- Verifying versions and integrity

💡 Example:

When you type `sudo apt install firefox`, the system:

1. Downloads the `firefox` package.
2. Resolves and downloads all dependencies.
3. Installs it.
4. Updates internal records (logs, index).

◆ 2. Installing, Updating, and Resolving Package Conflicts

📋 What does `apt update` do?

This doesn't upgrade software. It **updates the list** of available packages from online sources (repositories).

```
sudo apt update
```

Without it, your system might not know if updates are available.

⟳ `apt upgrade` vs `full-upgrade`

- `apt upgrade`: Upgrades installed packages **without removing any existing ones**.
- `apt full-upgrade`: Can **remove or replace packages** if necessary (use with caution).

✖ Dependency Conflicts

Sometimes, two packages require different versions of the same library – this creates a **conflict**.

🔧 Example:

```
sudo dpkg -i package.deb
```

You might get:

```
dependency error: libXYZ (>= 2.0) not found
```

To fix:

```
sudo apt install -f
```

This asks APT to **fix** broken dependencies.

◆ 3. Managing DEB, RPM, AppImage, Flatpak, Snap

🧱 DEB – Debian Package Format

- Used by: Debian, Ubuntu, Linux Mint
- Managed by: `apt` or `dpkg`

🧠 **APT** handles everything automatically (dependencies, upgrades).

🧠 **DPKG** is lower-level. You use it to **install a `.deb` file manually**.

🔧 Example:

```
sudo dpkg -i mypackage.deb
sudo apt install -f # to fix dependencies if needed
```

📋 Summary:

- `.deb` is efficient for stable distros
- Community has created massive repos (e.g., Ubuntu Universe)

📦 RPM – Red Hat Package Manager

- Used by: Red Hat, Fedora, CentOS, AlmaLinux
 - Managed by: `dnf`, `yum`, and `rpm`
- 🧠 **DNF/YUM** = handles dependencies
- 🧠 **RPM** = installs manually, doesn't resolve dependencies
- 🛠 Example:

```
sudo dnf install nginx
# or manually:
sudo rpm -ivh mypackage.rpm
```

📋 Summary:

- Very powerful, but historically more manual than APT
- Supports GPG signing, rollback, group installs

AppImage – Portable App

- Think of it as a **standalone executable**.
- You **don't install it**, you **run it directly**.

 Steps:

```
chmod +x AppImageName  
./AppImageName
```

 Summary:

- No root access needed
- Ideal for testing or carrying apps on USB
- Doesn't integrate well with system package managers

Flatpak – Universal Sandboxed App

- Cross-distro package format.
 - Runs in a sandboxed environment.
 - Each app includes **its own runtime**, isolated from the system.
-  You install a **runtime** (shared libraries) + the app itself.

 Example:

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

 Summary:

- Ideal for desktop GUI apps
- Uses **Flathub** as its default repository

Snap – Ubuntu's Self-Contained Package System

- Developed by Canonical (Ubuntu).
- Each Snap contains all dependencies.
- Runs in a container-like environment.

 Example:

```
sudo snap install vlc
snap list
```

 Summary:

- Popular in Ubuntu, slower startup time
- Easy auto-updates
- Does not depend on host distro

◆ 4. Installing Applications from Tarball Archives

A **tarball** is just a compressed archive:

- It can contain **binaries**, **source code**, or **portable versions** of software.
- `.tar.gz`, `.tgz`, `.tar.xz` are common formats.

 Example:

```
wget http://example.com/myapp.tar.gz
tar -xzf myapp.tar.gz
cd myapp
```

You might find a:

- `bin/` directory with an executable → Run directly
- `README.md` or `INSTALL` with manual instructions

 Summary:

- No automatic updates or dependency handling
- You manage everything manually

◆ 5. Compiling and Installing from Source

This is the **most advanced method**. You do it when:

- You need a specific version
- You want to enable/disable certain features
- You want to optimize for your hardware

🧱 Step-by-step: Compiling from Source

1. Install the build tools:

```
sudo apt install build-essential  
# Or on RHEL-based:  
sudo dnf groupinstall "Development Tools"
```

2. Download the source code:

```
wget https://example.org/tool.tar.gz  
tar -xzf tool.tar.gz  
cd tool
```

💡 Step-by-step: Compiling from Source

3. Configure the build:

```
./configure --prefix=/usr/local
```

This prepares the Makefiles and checks for dependencies.

4. Compile:

```
make
```

5. Install:

```
sudo make install
```

🔄 Optional cleanup:

```
make clean
```

✓ Conclusion

Format	Ideal For	Package Manager	Automatic Updates
DEB	Ubuntu/Debian servers	APT	✓
RPM	RHEL/CentOS/Fedora	DNF/YUM	✓
AppImage	Portable, no root	none	✗
Flatpak	Sandboxed GUI apps	Flatpak	✓
Snap	Ubuntu desktop/server	Snap	✓
Tarball	Custom builds	Manual	✗
Source	Advanced/Custom config	Manual	✗



DISK AND FILE SYSTEM MANAGEMENT

◆ 1. Overview of File System Types: **ext2**, **ext3**, **ext4**

A **file system** is a set of rules and data structures that control how data is stored and retrieved on a disk. It decides:

- How files and directories are organized
- How metadata (owner, date, permissions) is stored
- How data integrity is ensured

EXT Family: **ext2**, **ext3**, **ext4**

File System	Journaling	Max File Size	Performance	Reliability	Best Use Case
ext2	 No	~2 TB	Fastest	Unsafe if crash	USB sticks, read-only partitions
ext3	 Yes	~2 TB	Moderate	Good	Legacy systems needing journaling
ext4	 Yes	~16 TB	Best	Excellent	Default in modern Linux distros

- **ext4** supports extents (faster, less fragmentation), delayed allocation (performance), and large volumes.
- **ext2** is ideal for USB drives because it doesn't write extra journal data – reducing write cycles.
- **ext3** was the default before ext4; you may still see it in production systems.

🛠 Demo: Formatting Partitions

```
# Format as ext2  
sudo mkfs.ext2 /dev/sdb1  
  
# Format as ext3  
sudo mkfs.ext3 /dev/sdb1  
  
# Format as ext4  
sudo mkfs.ext4 /dev/sdb1
```

◆ 2. Partitioning and Logical Volume Management (LVM)

Partitioning divides a disk into logical sections. Each can have:

- A specific mount point (`/`, `/home`, `/var`)
- A different file system (`ext4`, `xfs`)
- Different roles (boot, swap, data)

Partition Table Types

- **MBR** (Legacy, max 2 TB, 4 primary partitions)
- **GPT** (Modern, required for UEFI, supports > 2 TB and many partitions)

✓ What is LVM (Logical Volume Manager)?

LVM is a storage layer **between physical disks and the file system**. It allows:

- Resizing partitions on the fly
- Snapshots for backup/testing
- Combining multiple physical disks into a single volume group

⌚ Why use LVM?

- You can extend `/home` without unmounting.
- You can move logical volumes between disks.
- You can easily create a backup snapshot before risky updates.

LVM Concepts

Layer	Command	Description
Physical Volume (PV)	<code>pvcreate</code>	Raw disk or partition
Volume Group (VG)	<code>vgcreate</code>	Pool of PVs
Logical Volume (LV)	<code>lvcreate</code>	Virtual partition from VG

🛠 Demo: Create a Logical Volume

```
# Step 1: Create two partitions (e.g., /dev/sdb1 and /dev/sdc1)
sudo pvcreate /dev/sdb1 /dev/sdc1

# Step 2: Create volume group
sudo vgcreate vgdata /dev/sdb1 /dev/sdc1

# Step 3: Create a logical volume
sudo lvcreate -L 5G -n lvbackups vgdata

# Step 4: Format it and mount
sudo mkfs.ext4 /dev/vgdata/lvbackups
sudo mkdir /mnt/backups
sudo mount /dev/vgdata/lvbackups /mnt/backups
```

◆ 3. Formatting and Mounting File Systems

Formatting prepares a device (partition or volume) by creating its internal file system structure (superblock, inodes, journal, etc.).

```
sudo mkfs.ext4 /dev/sdb1
```

Mounting attaches a file system to a directory (mount point), so that the OS can access its contents.

```
sudo mkdir /mnt/data  
sudo mount /dev/sdb1 /mnt/data
```

fstab and Permanent Mounting

The `/etc/fstab` file defines which partitions should be mounted automatically at boot.

Example entry:

```
UUID=b63f-44de /mnt/data ext4 defaults 0 2
```

To find the UUID of a device:

```
sudo blkid
```

To mount all defined in `fstab`:

```
sudo mount -a
```

◆ 4. Configuring Quotas for Users and Groups

A **quota** is a limit on disk usage for a **user** or **group** on a mounted file system.

You can limit:

- Disk space (blocks)
- Number of files (inodes)

🎯 Use Cases

- Prevent a single user from filling up the server
- Control storage usage in a shared environment
- Enforce fair usage among students or tenants

🧠 Required conditions

- The file system must be mounted with `usrquota` and/or `grpquota`
- Quotas must be initialized using `quotacheck`

🛠 Step-by-Step Setup

```
# 1. Enable quotas in fstab (ext4 only)
sudo nano /etc/fstab
/dev/sda1 /home ext4 defaults,usrquota,grpquota 0 2

# 2. Remount the filesystem
sudo mount -o remount /home

# 3. Create quota files and enable
sudo quotacheck -cug /home
sudo quotaon /home

# 4. Set quota for user alice
sudo edquota alice
```

🛠 Step-by-Step Setup

This opens an editor:

```
Disk quotas for user alice:  
/dev/sda1: blocks in use: 5000, soft limit: 8000, hard limit: 10000
```

To check quotas:

```
quota -u alice
```

◆ 5. Introduction to RAID

RAID stands for **Redundant Array of Independent Disks**. It's used to:

- Increase **reliability** (mirroring, parity)
- Improve **performance** (striping)
- Create **large volumes** from small disks

Software vs. Hardware RAID

- **Hardware RAID**: managed by a physical RAID controller
- **Software RAID**: configured by the OS (e.g., `mdadm` in Linux)

🎯 RAID Level Summary

RAID	Minimum Disks	Redundancy	Performance	Use Case
0	2	✗ None	✓ Fast	Scratch disks, temp data
1	2	✓ Mirror	Read: ✓	System disks, important data
5	3	✓ Parity	Balanced	NAS, backup
10	4	✓ Mirror+Stripe	✓✓ Fast + Safe	Databases

🛠 Create RAID 1 with `mdadm`

```
# Install tool
sudo apt install mdadm

# Create a RAID 1 mirror
sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdb /dev/sdc

# Format and mount
sudo mkfs.ext4 /dev/md0
sudo mount /dev/md0 /mnt/raidmirror

# View status
cat /proc/mdstat
```

Save config:

```
sudo mdadm --detail --scan >> /etc/mdadm/mdadm.conf
```



File and Access Management

◆ 1. Standard Permissions: SUID, SGID, Sticky Bit

Linux uses 3 permission categories:

- **User (u)** → the owner of the file
- **Group (g)** → users in the same group
- **Others (o)** → everyone else

Each can have **read (r)**, **write (w)**, and **execute (x)** permissions.

✓ What is SUID (Set User ID)?

SUID tells the system: **run this file with the permissions of the file's owner** (usually root), no matter who executes it.

💡 Use case:

- The `passwd` command lets regular users change their passwords, but needs root to edit `/etc/shadow`.

◆ 1. Standard Permissions: SUID, SGID, Sticky Bit

```
# Create a test script
sudo cp /usr/bin/passwd /tmp/passwdcopy
sudo chmod u+s /tmp/passwdcopy
ls -l /tmp/passwdcopy
```

Expected output:

```
-rwsr-xr-x 1 root root ... /tmp/passwdcopy
```

Note the `s` in place of the owner's `x`.

◆ 1. Standard Permissions: SUID, SGID, Sticky Bit

What is SGID (Set Group ID)?

SGID on files:

- The process runs with the group ownership of the file.

SGID on directories:

- **All new files and subdirectories inherit the directory's group**, instead of the user's primary group.

Use case:

- Shared development folders where all files must belong to group `devteam`.

◆ 1. Standard Permissions: SUID, SGID, Sticky Bit

```
sudo mkdir /srv/dev  
sudo chgrp devteam /srv/dev  
sudo chmod 2775 /srv/dev
```

Now any file created in `/srv/dev` will belong to group `devteam`.

What is the Sticky Bit?

Sticky bit on directories allows:

- **Only the file's owner (or root) to delete it**, even if others have write access.

Use case:

- Public shared folders like `/tmp`, where users can write but not delete each other's files.

◆ 1. Standard Permissions: SUID, SGID, Sticky Bit

```
sudo mkdir /srv/public  
sudo chmod 1777 /srv/public
```

Check with:

```
ls -ld /srv/public
```

Output:

```
drwxrwxrwt 2 root root 4096 ... /srv/public
```

Note the final `t` (sticky bit).

◆ 2. Configuring Access Control Lists (ACLs)

ACLs allow you to define **multiple permissions** per file or directory, for **specific users and groups**, beyond standard owner/group/other.

Standard permissions only allow:

- One owner
- One group

ACLs allow:

- Custom rules for users like bob: read-only, alice: full access.

🔧 Enable ACLs

Most modern Linux systems already support ACLs. To be sure:

1. The file system must be mounted with `acl` option (check `/etc/fstab`).
2. Install tools:

```
sudo apt install acl # Debian/Ubuntu  
sudo dnf install acl # RHEL/Fedora
```

🔧 Basic ACL Usage

```
# Set full access for alice
sudo setfacl -m u:alice:rwx /srv/data.txt

# Set read-only access for bob
sudo setfacl -m u:bob:r-- /srv/data.txt

# Set default ACL for a directory (inherit)
sudo setfacl -d -m u:alice:rwx /srv/shared

# View current ACLs
getfacl /srv/data.txt

# Remove ACL for bob
sudo setfacl -x u:bob /srv/data.txt

# Remove all ACLs
sudo setfacl -b /srv/data.txt
```

 **Example Test:**

```
# Step 1: Create a test file as root
sudo touch /srv/test.txt

# Step 2: Set permission to 600 (owner only)
sudo chmod 600 /srv/test.txt

# Step 3: Add read permission for bob
sudo setfacl -m u:bob:r-- /srv/test.txt

# Step 4: Switch to bob and try to read
sudo -u bob cat /srv/test.txt
```

If ACL is correctly set, bob will be able to read even though he is not the owner.

◆ 3. Setting Up Collaborative Directories

A shared folder where **multiple users can create/edit files**, with **group-based access control** and possibly **ACL inheritance**.

Use case:

In a software development team:

- Files in `/srv/dev` should always belong to the group `devteam`
- Only team members can write
- Files should inherit correct permissions and group

🛠 Setup Step-by-Step

```
# Create directory and group
sudo groupadd devteam
sudo mkdir /srv/dev

# Set group and permissions
sudo chgrp devteam /srv/dev
sudo chmod 2770 /srv/dev      # SGID bit ensures group inheritance

# Add users to devteam
sudo usermod -aG devteam alice
sudo usermod -aG devteam bob

# Optional: Set ACL defaults
sudo setfacl -d -m g:devteam:rwx /srv/dev
```

 **Test it:**

```
# Switch to alice (in devteam)
sudo -u alice touch /srv/dev/test1

# Switch to bob and check if he can edit
sudo -u bob echo "hello" > /srv/dev/test1
```

You should see that:

- File is owned by `alice:devteam`
- Bob can write because he's in the same group and permissions are inherited



DAEMON AND SERVICE MANAGEMENT

◆ 1. Understanding the Boot Process and Creating a Custom GRUB Configuration

GRUB2 (GRand Unified Bootloader version 2) is the program that loads your operating system **after the BIOS/UEFI finishes initializing hardware.**

It:

- Finds and loads the kernel (`vmlinuz`)
- Loads the initial RAM disk (`initrd` or `initramfs`)
- Passes parameters (like single user mode or runlevels)
- **Configuration files:**
 - `/etc/default/grub`: main settings (timeout, default OS, kernel parameters)
 - `/etc/grub.d/`: scripts used to build the boot menu
 - `/boot/grub/grub.cfg`: final bootloader configuration (auto-generated!)

◆ 1. Understanding the Boot Process and Creating a Custom GRUB Configuration

1. Edit default config:

```
sudo nano /etc/default/grub
```

Important variables:

```
GRUB_DEFAULT=0                      # First menu entry
GRUB_TIMEOUT=5                       # Seconds before boot
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"  # Kernel options (ex: `init=/bin/bash`)
```

2. Generate updated configuration:

```
# Debian/Ubuntu
sudo update-grub

# RHEL/Fedora
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

◆ 1. Understanding the Boot Process and Creating a Custom GRUB Configuration

 **Example: Add a custom GRUB menu entry**

```
sudo nano /etc/grub.d/40_custom
```

Add this entry:

```
menuentry "Rescue Shell" {
    set root=(hd0,1)
    linux /vmlinuz root=/dev/sda1 ro init=/bin/bash
    initrd /initrd.img
}
```

Then regenerate:

```
sudo update-grub
```

Use this entry at boot to launch into a shell without full init.

◆ 2. How **systemd** Works - Units, Services, and Targets

systemd is the init system and service manager for most modern Linux distributions. It replaces:

- SysVinit (`/etc/init.d/`)
- Upstart (Ubuntu before 15.04)

It manages:

- Services (`*.service`)
- Mount points (`*.mount`)
- Devices (`*.device`)
- Boot targets (`*.target`)
- Timers, sockets, etc.

Key Concepts

Unit Type	Description	Example
.service	Start, stop, restart processes	apache2.service
.target	Group of units (like runlevel)	multi-user.target
.timer	Schedule tasks (cron alternative)	logrotate.timer
.socket	Activates services on socket event	cups.socket

◆ 3. Creating and Managing Services with `systemd`

1. Create your script:

```
sudo nano /usr/local/bin/myservice.sh
```

```
#!/bin/bash
echo "$(date): Service started" >> /var/log/myservice.log
```

2. Make it executable:

```
sudo chmod +x /usr/local/bin/myservice.sh
```

3. Create the service file:

```
sudo nano /etc/systemd/system/myservice.service
```

◆ 3. Creating and Managing Services with `systemd`

```
[Unit]
Description=My Custom Logging Service
After=network.target

[Service]
ExecStart=/usr/local/bin/myservice.sh
Type=oneshot

[Install]
WantedBy=multi-user.target
```

4. Enable and start it:

```
sudo systemctl daemon-reexec      # Reload binary
sudo systemctl daemon-reload       # Reload unit files
sudo systemctl enable myservice
sudo systemctl start myservice
```

◆ 4. Creating Custom Targets in `systemd`

A **target** is a logical group of systemd units (like a runlevel). You can:

- Start a target (set of services)
- Make it default
- Create your own for specific use cases

Create a Custom Target

1. Create the target unit:

```
sudo nano /etc/systemd/system/maintenance.target
```

```
[Unit]
Description=Maintenance Target
Requires=myservice.service
```

◆ 4. Creating Custom Targets in `systemd`

2. Enable the target:

```
sudo systemctl daemon-reload  
sudo systemctl isolate maintenance.target
```

3. Make it default (optional):

```
sudo systemctl set-default maintenance.target
```

To revert:

```
sudo systemctl set-default graphical.target
```

◆ 5. Managing SysVinit Services

Legacy software or old systems still use **SysVinit scripts** in `/etc/init.d`.

You can still manage them with `systemctl`, thanks to compatibility layers.

🔧 Examples

```
# Start legacy service
sudo /etc/init.d/ssh start

# Or use systemctl
sudo systemctl start ssh

# Enable SysV service at boot
sudo update-rc.d ssh enable      # Debian
sudo chkconfig ssh on            # RHEL
```

Check if you have `/etc/init.d/service` scripts:

```
ls /etc/init.d/
```

◆ 6. Rescue and Emergency Modes – Real Use

✓ What are they?

Mode	Description
rescue.target	Minimal environment, with root shell and basic services
emergency.target	Only root shell, no services, no mounts

🎯 Why use them?

- Fix corrupted `/etc/fstab`
- Reset forgotten root password
- Disable a faulty service causing boot failure

 **Access these modes:**

Option 1: From GRUB

1. At boot, press `e` on your GRUB entry
2. Find the `linux` line and add:

```
systemd.unit=rescue.target
```

or:

```
systemd.unit=emergency.target
```

3. Press `F10` or `Ctrl+X` to boot

**Option 2: From a running system

**

```
sudo systemctl isolate rescue.target  
sudo systemctl isolate emergency.target
```

To return:

```
sudo systemctl default
```

 **Example: Recover from broken /etc/fstab**

1. Add fake line:

```
echo "/dev/fake /mnt/fake ext4 defaults 0 2" | sudo tee -a /etc/fstab
```

2. Reboot → system hangs

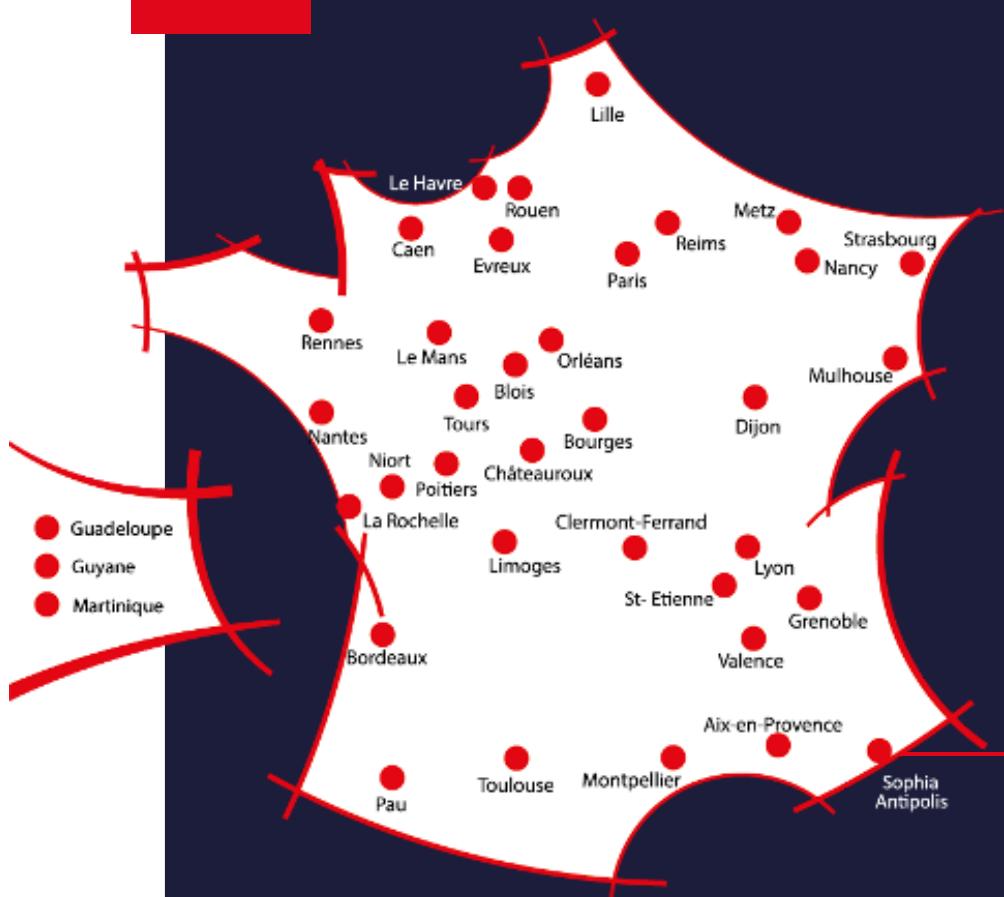
3. At GRUB, enter:

```
systemd.unit=emergency.target
```

4. Remount root:

```
mount -o remount,rw /
nano /etc/fstab
```

5. Fix the line, save, and reboot.



Découvrez également
l'ensemble des stages à votre disposition
sur notre site m2iformation.fr

m2iformation.fr

