Devops Shack

Managing Packages and Processes in Linux

By DevOps Shack

[Click here for DevSecOps & Cloud DevOps Course]

DevOps Shack

Table of Contents

- Killing/stopping/resuming processes (kill, nice, renice)

8. **Automating and Managing Startup Services**
    - Using systemd, service, and init.d
    - Enabling/disabling services on boot

1. Introduction

1.1 What is Package Management?

Package management refers to a system of tools and processes that simplify the installation, upgrade, configuration, and removal of software applications and libraries in Linux systems. It ensures software components are correctly installed, updated, and maintained with minimal user intervention.

1.2 What is a Package?

A **package** is an archive file containing:

- Compiled software or binaries
- Metadata (name, version, description, dependencies)
- Configuration files
- Scripts (for installation or removal)

Packages usually have extensions like .deb (Debian-based), .rpm (Red Hat-based), or are delivered in universal formats like Snap, Flatpak, or AppImage.

1.3 Key Roles of Package Managers

Package managers help users:

- Search for available software
- Install and remove software cleanly
- Automatically resolve and install dependencies
- Upgrade packages to newer versions
- Maintain consistency and reduce software conflicts

1.4 Types of Package Management Systems

Linux distributions typically use one of the following two types of package managers:

a) Binary Package Managers

These handle precompiled binaries (ready-to-install software):

- apt (Advanced Package Tool) – Ubuntu/Debian

- yum / dnf – Red Hat, CentOS, Fedora
- zypper – openSUSE
- pacman – Arch Linux

b) Source-based Package Managers

These compile packages from source:

- portage (Gentoo)
- pkgsrc (NetBSD)

1.5 Centralized vs. Decentralized Systems

- **Centralized**: Packages come from official repositories (e.g., Ubuntu's main repo)
- **Decentralized**: Packages are installed from multiple sources (e.g., Snapcraft, Flatpak, PPAs)

1.6 Benefits of Package Management

- **Efficiency**: Avoids manual compilation and installation
- **Security**: Signed packages and regular updates
- **Scalability**: Easily scriptable for enterprise-wide deployments
- **Dependency Handling**: Automatically installs required libraries and tools

1.7 Components of a Package Management System

- **Package manager tool**: e.g., apt, yum, pacman
- **Repositories**: Server locations that store and serve packages
- **Package database**: Keeps track of installed packages on a system

1.8 Summary

Understanding package management is foundational for working efficiently in Linux. Whether you're deploying a simple tool or managing hundreds of servers, knowing how to handle packages ensures a stable and maintainable environment.

2. Popular Package Managers

Linux distributions rely on different package managers depending on their family (Debian-based, Red Hat-based, etc.). Each package manager offers commands and syntax to handle software efficiently.

2.1 APT (Advanced Package Tool) – Debian, Ubuntu

File Extension: .deb
Common Commands:

```
sudo apt update              # Update package index
sudo apt install nginx       # Install a package
sudo apt upgrade             # Upgrade all installed packages
sudo apt remove nginx        # Remove a package
sudo apt purge nginx         # Remove a package with configuration
sudo apt autoremove          # Remove unused dependencies
```

Features:

- Automatically resolves dependencies
- Uses /etc/apt/sources.list for repository configurations
- Easy to use and widely documented

2.2 DNF/YUM – RHEL, CentOS, Fedora

File Extension: .rpm
DNF is the next-generation replacement for **YUM**.
Common Commands (DNF):

```
sudo dnf install httpd       # Install package
sudo dnf remove httpd        # Remove package
sudo dnf update              # Update all packages
sudo dnf search nginx        # Search for packages
sudo dnf info nginx          # Show package details
```

YUM is still found in older CentOS/RHEL systems.

Features:

- Handles group installs (e.g., sudo dnf groupinstall "Development Tools")
- Built-in support for plugins
- Better performance and metadata handling in DNF over YUM

2.3 Zypper – openSUSE

File Extension: .rpm
Common Commands:

```
sudo zypper refresh          # Refresh repository metadata
sudo zypper install apache2   # Install package
sudo zypper remove apache2    # Remove package
sudo zypper update           # Update system
```

Features:

- Fast, simple, interactive interface
- Good support for system snapshots using Btrfs and Snapper

2.4 Pacman – Arch Linux, Manjaro

File Extension: .pkg.tar.zst
Common Commands:

```
sudo pacman -Syu             # Sync and upgrade all
sudo pacman -S firefox       # Install package
sudo pacman -R firefox       # Remove package
sudo pacman -Ss apache       # Search packages
```

Features:

- Very fast and lightweight
- Manages both local and remote packages

- Allows installing from AUR (Arch User Repository) via helpers like yay

2.5 Snap – Universal Package Manager (Canonical)

File Extension: .snap
Common Commands:

```
sudo snap install code        # Install VSCode snap
sudo snap list                # List installed snaps
sudo snap remove code         # Remove snap
```

Features:

- Sandboxed applications
- Auto-updates
- Cross-distro compatibility

2.6 Flatpak – Universal Linux App Distribution

Common Commands:

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

Features:

- Designed for desktop apps
- Works across distributions
- Isolated runtime environments

2.7 AppImage – Portable Executables for Linux

Features:

- No installation needed; just download and run
- No dependency management

- Ideal for single-use or portable tools

2.8 Summary Table

Package Manager	Distros	File Type	GUI Frontends
APT	Debian, Ubuntu	.deb	Synaptic, Software Center
DNF/YUM	RHEL, Fedora	.rpm	Dnfdragora
Zypper	openSUSE	.rpm	YaST
Pacman	Arch, Manjaro	.pkg.tar.zst	Pamac
Snap	All (Ubuntu-led)	.snap	Snap Store
Flatpak	All (desktop)	-	GNOME Software
AppImage	All	.AppImage	AppImageLauncher

3. Installing, Updating, and Removing Packages in Linux

One of the most routine and vital tasks for Linux users and administrators is managing software packages—installing new software, updating existing ones, and removing unused applications. The commands and processes vary slightly depending on the package manager and Linux distribution.

3.1 Installing Packages

Using APT (Debian/Ubuntu)

```
sudo apt install <package-name>

# Example:

sudo apt install nginx
```

Using DNF (RHEL/Fedora)

```
sudo dnf install <package-name>

# Example:

sudo dnf install httpd
```

Using YUM (older RHEL/CentOS)

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

Using Pacman (Arch/Manjaro)

```
sudo pacman -S <package-name>

# Example:

sudo pacman -S firefox
```

Using Zypper (openSUSE)

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

Using Snap

```
sudo snap install <package-name>

# Example:

sudo snap install code
```

Using Flatpak

flatpak install flathub <app-id>

Example:

flatpak install flathub org.gimp.GIMP

3.2 Updating Packages

APT

sudo apt update # Refresh package lists

sudo apt upgrade # Upgrade all packages

DNF

sudo dnf upgrade # Upgrade all packages

Pacman

sudo pacman -Syu # Sync repos and upgrade system

Zypper

sudo zypper update

Snap

sudo snap refresh # Updates all installed snaps

Flatpak

flatpak update

3.3 Removing or Uninstalling Packages

APT

sudo apt remove <package-name> # Removes package, keeps config

sudo apt purge <package-name> # Removes package and config

DNF

sudo dnf remove <package-name>

Pacman

```
sudo pacman -R <package-name>       # Remove package only
sudo pacman -Rs <package-name>      # Remove with unused dependencies
```

Zypper

```
sudo zypper remove <package-name>
```

Snap

```
sudo snap remove <package-name>
```

Flatpak

```
flatpak uninstall <app-id>
```

3.4 Autoremove and Cleaning Up

Sometimes, package installations pull in dependencies that are no longer required.

- **APT:**

```
sudo apt autoremove        # Remove unused packages
sudo apt clean             # Clear local repository cache
```

- **DNF:**

```
sudo dnf autoremove
```

- **Pacman:**

```
sudo pacman -Rns $(pacman -Qdtq)   # Remove orphaned packages
```

3.5 Dealing with Broken or Failed Installations

APT

```
sudo apt --fix-broken install      # Fix dependency issues
sudo dpkg --configure -a           # Reconfigure half-installed packages
```

DNF

DNF automatically resolves conflicts, but logs and dnf history can help identify issues:

```
sudo dnf history rollback <ID>      # Roll back to a previous state
```

3.6 Installing from .deb or .rpm Manually

.deb File

```
sudo dpkg -i package.deb
sudo apt install -f     # Fix dependencies
```

.rpm File

```
sudo rpm -ivh package.rpm
```

For safer handling with dependency resolution, prefer:

```
sudo dnf install package.rpm
```

3.7 Summary Checklist

Action	APT (Debian)	DNF (RHEL)	Pacman (Arch)	Snap/Flatpak
Install	apt install	dnf install	pacman -S	snap install / flatpak install
Update	apt upgrade	dnf upgrade	pacman -Syu	snap refresh / flatpak update
Remove	apt remove/purge	dnf remove	pacman -R/-Rs	snap remove / flatpak uninstall
Cleanup	apt autoremove	dnf autoremove	pacman -Rns	-

4. Searching and Querying Packages

Efficient package management requires the ability to **find**, **inspect**, and **verify** packages—both before installation and after they're on the system. Each package manager provides a set of commands to help search repositories, view details, and list package contents or dependencies.

4.1 Searching for Packages

APT (Debian/Ubuntu)

```
apt search <package-name>
# Example:
apt search nginx
```

Or use apt-cache (older utility):

```
apt-cache search nginx
```

DNF/YUM (RHEL/Fedora)

```
dnf search <package-name>
```

Example:

```
dnf search php
```

Pacman (Arch/Manjaro)

```
pacman -Ss <search-term>
# Example:
pacman -Ss nginx
```

Zypper (openSUSE)

```
zypper search <package-name>
```

Example:

```
zypper search apache
```

Snap

```
snap find <package-name>
```

Example:

snap find vlc

Flatpak

flatpak search <package-name>

Example:

flatpak search gimp

4.2 Querying Package Details

To check a package's metadata such as version, description, dependencies, and repository source.

APT

apt show <package-name>

or

dpkg -s <package-name> # For installed packages

DNF

dnf info <package-name>

Pacman

pacman -Si <package-name> # Remote repo info

pacman -Qi <package-name> # Installed package info

Zypper

zypper info <package-name>

Snap

snap info <package-name>

Flatpak

flatpak info <package-name>

4.3 Listing Installed Packages

APT

dpkg -l # List all installed packages

apt list --installed # Better format

DNF/YUM

dnf list installed

Pacman

pacman -Q # List all installed packages

Zypper

zypper se --installed-only

Snap

snap list

Flatpak

flatpak list

4.4 Viewing Package Files and Locations

To know what files are part of a package and where they are installed.

APT

dpkg -L <installed-package>

Or find which package installed a specific file

dpkg -S /usr/bin/nginx

DNF

dnf repoquery -l <package-name> # Requires `dnf-plugins-core`

rpm -ql <installed-package>

Pacman

pacman -Ql <package-name>

Zypper

rpm -ql <package-name>

4.5 Checking Dependencies

APT

apt depends <package-name> # List direct dependencies

apt rdepends <package-name> # List reverse dependencies

DNF

dnf deplist <package-name>

Pacman

pactree <package-name> # Hierarchical view

4.6 Verifying Package Integrity

APT (with dpkg)

debsums <package-name> # Check file integrity (must install `debsums`)

DNF/RPM

rpm -V <package-name>

Pacman

pacman -Qk <package-name> # Verify files

4.7 Summary Table

Task	APT	DNF/YUM	Pacman	Snap	Flatpak
Search packages	apt search	dnf search	pacman -Ss	snap find	flatpak search
View details	apt show	dnf info	pacman -Si	snap info	flatpak info
List installed	dpkg -l	dnf list	pacman -	snap list	flatpak list

Task	APT	DNF/YUM	Pacman	Snap	Flatpak
packages		installed	Q		
List package files	dpkg -L	rpm -ql	pacman -Ql	-	-
Check dependencies	apt depends	dnf deplist	pactree	-	-

5. Managing Software Repositories

In Linux, **repositories** (repos) are centralized sources where packages are stored and maintained. Managing these repositories allows users to control which software sources their system trusts and uses, add third-party tools, and optimize system updates.

5.1 What Is a Software Repository?

A **repository** is a server or a local directory containing a collection of software packages and metadata. Most Linux distributions come with a set of **default repositories**, but users can add third-party or custom repositories for more software options.

Repositories are typically defined in:

- **APT (Debian/Ubuntu):** /etc/apt/sources.list and /etc/apt/sources.list.d/*.list
- **DNF/YUM (RHEL/Fedora):** /etc/yum.repos.d/*.repo
- **Pacman (Arch):** /etc/pacman.conf
- **Zypper (openSUSE):** /etc/zypp/repos.d/*.repo

5.2 Viewing Current Repositories

APT

cat /etc/apt/sources.list

ls /etc/apt/sources.list.d/

DNF/YUM

dnf repolist

yum repolist

Pacman

cat /etc/pacman.conf

Zypper

zypper repos

5.3 Adding New Repositories

APT

Use add-apt-repository or manually edit .list files:

sudo add-apt-repository ppa:graphics-drivers/ppa

sudo apt update

Or manually:

echo "deb http://repository.url/ubuntu focal main" | sudo tee /etc/apt/sources.list.d/custom.list

To add GPG keys:

wget -qO - https://repo.key.url | sudo apt-key add -

DNF/YUM

Create a .repo file:

```
# /etc/yum.repos.d/custom.repo
[custom-repo]
name=Custom Repo
baseurl=http://repository.url/path
enabled=1
gpgcheck=1
gpgkey=http://repository.url/RPM-GPG-KEY-custom
```

Or install repo packages directly:

```
sudo dnf install https://repo.url/package-release.rpm
```

Pacman

Edit /etc/pacman.conf:

```
[customrepo]
SigLevel = Optional TrustAll
Server = http://repo.url/path
```

Then update:

```
sudo pacman -Sy
```

Zypper

```
sudo zypper addrepo http://download.opensuse.org/repositories/home:/repo.repo
sudo zypper refresh
```

5.4 Removing or Disabling Repositories

APT

Remove or comment out entries in .list files:

```
sudo rm /etc/apt/sources.list.d/custom.list
sudo apt update
```

DNF

```
sudo dnf config-manager --disable repo-id
# Or delete the .repo file manually:
sudo rm /etc/yum.repos.d/custom.repo
```

Pacman

Edit /etc/pacman.conf and comment out the section.

Zypper

```
sudo zypper removerepo <repo-alias>
```

5.5 Priority and Pinning (Advanced)

To prioritize one repo over another or restrict package versions.

APT - Pinning

Use /etc/apt/preferences.d/*.pref files:

```
Package: *
Pin: origin "ppa.launchpad.net"
Pin-Priority: 700
```

DNF - Priorities

Install plugin and configure:

```
sudo dnf install 'dnf-plugins-core'
```

In .repo file:

```
priority=10
```

Lower numbers = higher priority.

5.6 Signing Keys and Security

Repository signing ensures packages are from a trusted source.

- **APT:** Uses apt-key or gpg
- **DNF:** Uses gpgkey in .repo file
- **Pacman:** Uses GPG keyring (pacman-key)
- **Zypper:** Asks for confirmation to trust new keys

Always verify that you're adding keys from legitimate sources.

5.7 Repository Mirrors

To improve download speeds, use geographically closer mirrors.

APT

sudo sed -i 's|http://archive.ubuntu.com|http://mirror.provider.com|' /etc/apt/sources.list

Pacman

sudo pacman-mirrors --fasttrack

Zypper

zypper modifyrepo --priority 90 repo-name

5.8 Summary Table

Task	APT	DNF/YUM	Pacman	Zypper
View repos	cat sources.list	dnf repolist	cat pacman.conf	zypper repos
Add repo	add-apt-repository	.repo file or dnf install	Edit pacman.conf	zypper addrepo
Remove repo	Delete .list file	Remove .repo file	Edit pacman.conf	zypper removerepo
Set priority	apt pinning	priority=X in repo file	Manual order	--priority flag
Add GPG key	apt-key add	gpgkey= line in .repo	pacman-key	Prompted by Zypper

6. Process Management Basics

In Linux, a **process** is an instance of a running program. Every time a command or program is executed, the system creates a new process. Understanding and managing these processes is vital for maintaining system stability, performance, and responsiveness.

6.1 What is a Process in Linux?

A **process** includes:

- Program code (text)
- Program counter (instruction pointer)
- Stack (function parameters, return addresses)
- Data section (variables)
- File descriptors and environment variables

Each process is identified by a unique **Process ID (PID)**.

6.2 Process Lifecycle

1. **Creation** – Initiated by system calls like fork() or exec().
2. **Ready/Waiting** – Waiting for CPU time or input.
3. **Running** – Currently being executed.
4. **Sleeping** – Waiting for events (I/O, signals).
5. **Stopped/Zombie** – Suspended or terminated, but not cleaned up.

6.3 Viewing Active Processes

ps (Process Status)

Shows snapshot of processes.

```
ps aux          # All processes
ps -ef          # Full-format listing
ps -u <user>    # Processes by user
```

top

Interactive, real-time process monitor.

```
top
```

Keys:

- P – Sort by CPU
- M – Sort by memory
- k – Kill a process
- q – Quit

htop *(Enhanced top, needs installation)*

```
htop
```

Features:

- Colorful display
- Easy sorting and filtering
- Mouse support

pidof

Get PID of a running program.

```
pidof apache2
```

pgrep

Find processes by name.

```
pgrep firefox
```

6.4 Foreground and Background Processes

Run in background

```
./script.sh &
```

Bring process to background/foreground

```
jobs                # List background jobs
```

```
fg %1              # Bring job 1 to foreground
bg %1              # Resume job 1 in background
```

6.5 Managing Process Priorities (Nice & Renice)

The **nice value** controls process priority (range: -20 to 19; lower is higher priority).

Start a process with nice

```
nice -n 10 ./script.sh
```

Change priority of running process

```
renice -n 5 -p <PID>
```

6.6 Killing/Stopping Processes

kill

Sends signal to a process.

```
kill <PID>          # Default SIGTERM (15)
kill -9 <PID>       # SIGKILL (force)
```

killall

Kill by name.

```
killall firefox
```

pkill

Pattern-based kill.

```
pkill -f nginx
```

6.7 Process Trees

pstree

Displays process hierarchy.

```
pstree -p          # Show PIDs
```

6.8 Daemons and Background Services

Daemons are background processes often started at boot (e.g., sshd, cron).

Use systemctl or service to manage:

sudo systemctl status apache2

sudo systemctl start|stop|restart|enable apache2

6.9 Signals (Commonly Used)

Signal	Name	Description
1	SIGHUP	Reload config
9	SIGKILL	Force kill
15	SIGTERM	Graceful termination
18	SIGCONT	Continue
19	SIGSTOP	Pause/Stop

Use kill -l to list all signals.

6.10 Summary Table

Command	Description
ps aux	Snapshot of all processes
top, htop	Real-time process monitoring
jobs, fg, bg	Foreground/background job control
nice, renice	Set/change process priority
kill, killall, pkill	Terminate processes
pstree	Visualize parent-child process structure

7. Monitoring and Controlling Processes

Monitoring and controlling processes in Linux ensures efficient system performance, prevents resource hogging, and helps with debugging and system

troubleshooting. This involves real-time observation, resource usage tracking, and reactive control techniques.

7.1 Real-Time Process Monitoring Tools

top

```
top
```

- Displays processes sorted by CPU/memory usage.
- Use keys like P, M, k, q to interact.

htop

```
htop
```

- Enhanced visual display.
- Tree view, color coding, interactive selection.
- Shows CPU cores, memory usage, and load average.

glances

```
glances
```

- Cross-platform system monitoring.
- Summarizes CPU, memory, I/O, network, disk, and sensors.

Install with: `sudo apt install glances` or `pip install glances`

7.2 Resource Usage Monitoring

CPU Usage

```
mpstat -P ALL 1
```

Memory Usage

```
free -h
```

Disk Usage

```
df -h          # Filesystem usage
du -sh *       # Size of directories
```

Network Usage

nload # Real-time bandwidth

iftop # Per-process network usage

7.3 Checking Process Resource Usage

pidstat

pidstat -p <PID>

Displays per-process CPU, memory, I/O stats.

time

time ./script.sh

Shows time taken and system resources used by a command.

/proc filesystem

A virtual filesystem for process info:

cat /proc/<PID>/status

cat /proc/<PID>/cmdline

cat /proc/<PID>/fd

7.4 Tracing and Debugging Processes

strace

Traces system calls and signals of a process.

strace -p <PID>

lsof (List Open Files)

lsof -p <PID> # Open files by process

lsof -i :80 # Processes using port 80

gdb (GNU Debugger)

Attach debugger to a running process:

```
gdb -p <PID>
```

7.5 Controlling Process Behavior

Renicing Running Process

```
renice 10 -p <PID>
```

Pausing and Resuming

```
kill -STOP <PID>      # Pause
kill -CONT <PID>      # Resume
```

Limiting Resources with ulimit

Set limits on processes (shell level):

```
ulimit -u 100      # Max user processes
ulimit -n 4096     # Max open files
```

Using cgroups (Advanced)

Control CPU/memory/network per process group:

```
# Example with systemd-run
systemd-run --scope -p MemoryMax=100M ./script.sh
```

7.6 Log Monitoring for Process Info

journalctl

For systemd-based services:

```
journalctl -u apache2.service
```

/var/log/

- /var/log/syslog or /var/log/messages for system events.
- /var/log/dmesg for kernel ring buffer.
- Application-specific logs.

7.7 GUI Tools for Monitoring (Optional)

- **System Monitor (GNOME/KDE)**
- **KSysGuard**
- **Xfce Task Manager**

Install via your desktop environment's package manager.

7.8 Summary Table

Tool/Command	Function
top, htop	Real-time CPU/memory/process stats
glances	Overall system summary
pidstat, time	Per-process resource usage
strace	System call tracing
lsof	View open files and ports
kill -STOP/-CONT	Pause/resume processes
ulimit	Limit per-user resource consumption
systemd-run	Run commands with resource limits

8. Best Practices for Package and Process Management

Managing packages and processes effectively is essential for maintaining the stability, performance, and security of any Linux system—especially in multi-

user or production environments. This section outlines best practices to help you stay in control and minimize risks.

8.1 Package Management Best Practices

Use Official Repositories

- Stick to the distro's official or verified repositories unless absolutely necessary.
- Avoid third-party or unsigned packages to reduce the risk of malware or dependency issues.

Keep the System Updated

- Regular updates patch vulnerabilities and bring performance improvements.
- Automate updates for security patches (e.g., unattended-upgrades in Debian-based systems).

```
# Ubuntu/Debian
sudo apt update && sudo apt upgrade -y

# CentOS/RHEL
sudo dnf update -y
```

Use Package Signing and Verification

- Ensure packages are signed with trusted GPG keys.
- Example: apt-key, rpm --checksig, or dnf check-update.

Avoid Manual Binary Installations

- Installing software by downloading and extracting .tar.gz files or precompiled binaries may bypass dependency and update management.

Remove Unused Packages

- Clear out unnecessary packages to free up space and reduce attack surface.

```
sudo apt autoremove

sudo dnf autoremove
```

8.2 Process Management Best Practices

Monitor Regularly

- Use top, htop, or glances in cron jobs or dashboard tools for continuous monitoring.
- Set up alerts for high CPU or memory usage.

Avoid Zombie Processes

- Always wait() on child processes in custom scripts.
- Monitor for zombies using ps aux | grep Z.

Limit Resource Consumption

- Use ulimit, cgroups, or systemd resource directives to limit CPU, memory, or file descriptor usage.

Use Systemd Services Properly

- Run long-lived or background tasks as systemd services with proper configuration:

```
[Service]

Restart=on-failure

MemoryLimit=200M

CPUQuota=50%
```

Graceful Process Handling

- Use SIGTERM instead of SIGKILL where possible to allow cleanup.
- Trap signals in scripts to handle shutdown gracefully.

```
trap "echo Shutting down; exit" SIGINT SIGTERM
```

Use Logging and Auditing

- Ensure processes log errors and actions to a file or syslog.

- Monitor logs using tools like logrotate, journalctl, or fail2ban.

8.3 Security Considerations

- **Run processes with least privilege** — Avoid running apps as root unless necessary.
- **Use chroot or containers** for isolation.
- **Scan for rootkits and malware** using tools like rkhunter, chkrootkit.

8.4 Automation & Configuration Management

- Use **Ansible**, **Puppet**, or **Chef** to automate package installations and process configurations.
- Maintain **version-controlled** configuration files (/etc) using Git or etckeeper.

8.5 Documentation and Audits

- Document all installed packages and running services.
- Periodically audit:
    - Installed packages
    - Active processes and services
    - Resource usage trends

dpkg --get-selections > installed_packages.txt

systemctl list-units --type=service > active_services.txt

8.6 Summary Checklist

Area	Best Practices
Packages	Use official repos, verify signatures, keep updated, clean unused
Processes	Monitor usage, avoid zombies, limit resources, use systemd

Area	Best Practices
Security	Drop privileges, use logging, audit regularly
Automation	Use CM tools (Ansible/Puppet), document setups

✅ Conclusion

Understanding and mastering **package and process management** in Linux is fundamental for any system administrator, DevOps engineer, or advanced user. Following best practices ensures you have:

- A secure and stable system
- Efficient performance
- Controlled and accountable resource usage
- Automated, documented, and reproducible setups