

Package Management

LinuXpert Systems

No.5, Dr. Ambedkar St, Rathinapuram,

Nanganallur, Chennai – 600 061.

Phone: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. apt 1

2. dpkg 9

3. dnf 16

4. rpm 24

5. pacman 32

6. zypper 40

1.apt

The **apt (Advanced Package Tool)** command is the standard package management utility used on **Debian-based Linux distributions** such as:

- Ubuntu
- Debian
- Linux Mint
- Pop!_OS
- Kali Linux

It is used to **install, update, upgrade, remove, and manage software packages** from software repositories.

Syntax

```
apt [options] command [package_name]
```

Example:

```
sudo apt install nginx
```

Why use apt?

Instead of manually downloading software and resolving dependencies, apt:

- Downloads packages
- Installs dependencies automatically
- Keeps software updated
- Removes unused packages
- Searches available software
- Verifies package integrity

Common apt Commands

1. Update Package List

```
sudo apt update
```

Purpose

Downloads the latest package information from configured repositories.

Example Output

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

This **does not install updates**. It only refreshes the package database.

---

2. Upgrade Installed Packages

```
sudo apt upgrade
```

Upgrades all installed packages to their newest available versions without removing packages.

Example:

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

---

3. Full Upgrade

```
sudo apt full-upgrade
```

Also upgrades packages but can:

- Install new dependencies
- Remove conflicting packages

Useful during major system upgrades.

4. Install a Package

```
sudo apt install package_name
```

Example

```
sudo apt install git
```

Installs Git and any required dependencies.

---

5. Install Multiple Packages

```
sudo apt install git vim curl
```

Installs all three packages together.

---

6. Remove a Package

```
sudo apt remove package_name
```

Example

```
sudo apt remove apache2
```

Removes the program but leaves configuration files.

---

7. Remove Package Completely

```
sudo apt purge package_name
```

Example

```
sudo apt purge apache2
```

Removes:

- Program
- Configuration files

8. Automatically Remove Unused Packages

```
sudo apt autoremove
```

Removes unnecessary dependency packages that are no longer needed.

---

9. Clean Downloaded Package Cache

```
sudo apt clean
```

Deletes all downloaded package files stored in:

```
/var/cache/apt/archives/
```

Useful when freeing disk space.

---

10. Search for Packages

```
apt search package_name
```

Example

```
apt search firefox
```

Output

```
firefox
firefox-locale-en
firefox-esr
```

---

11. Show Package Information

```
apt show package_name
```

Example

```
apt show nginx
```

Displays:

- Version
- Dependencies
- Description
- Installed size
- Homepage

---

12. List Installed Packages

```
apt list --installed
```

Example

```
apt list --installed
```

---

13. List Upgradable Packages

```
apt list --upgradable
```

Shows packages that have newer versions available.

---

14. Reinstall a Package

```
sudo apt install --reinstall package_name
```

Example

```
sudo apt install --reinstall openssh-server
```

Useful if package files become corrupted.

---

15. Download a Package Without Installing

```
apt download package_name
```

Example

```
apt download nginx
```

Downloads the `.deb` package into the current directory.

Frequently Used Workflow

Update package list

```
sudo apt update
```

↓

Upgrade installed software

```
sudo apt upgrade
```

↓

Remove unused packages

```
sudo apt autoremove
```

↓

Clean cache

```
sudo apt clean
```

Examples

Install VLC

```
sudo apt install vlc
```

Install Git

```
sudo apt install git
```

Remove Git

```
sudo apt remove git
```

Search for Docker

```
apt search docker
```

View package details

```
apt show docker.io
```

Check available updates

```
apt list --upgradable
```

Important Files Used by `apt`

File/Directory	Purpose
`/etc/apt/sources.list`	Main repository list
`/etc/apt/sources.list.d/`	Additional repository files
`/var/cache/apt/archives/`	Downloaded package cache
`/var/lib/apt/lists/`	Downloaded package indexes

Difference Between `apt`, `apt-get`, and `dpkg`

Command	Purpose
`apt`	User-friendly package management interface (recommended for interactive use)
`apt-get`	Lower-level package management tool, commonly used in scripts
`dpkg`	Installs, removes, and queries local `.deb` packages; does not automatically resolve dependencies

Exit Status

- **0** – Command completed successfully.
- **Non-zero** – An error occurred (e.g., package not found, repository unavailable, insufficient permissions).

Best Practices

- Run `sudo apt update` before installing or upgrading packages.
- Use `sudo apt upgrade` regularly to keep your system secure.
- Use `sudo apt autoremove` periodically to clean unused dependencies.
- Use `sudo apt purge` if you want to completely remove a package and its configuration files.
- Avoid interrupting `apt` while it is installing or upgrading packages, as this may leave the package database in an inconsistent state.

`apt` is one of the most essential commands on Debian-based Linux systems, providing a simple interface for managing software throughout its lifecycle—from installation and updates to removal and maintenance.

2. dpkg

The **dpkg (Debian Package)** command is a low-level package management tool used to **install, remove, configure, and manage `.deb` packages** on Debian-based Linux distributions such as:

- Ubuntu
- Debian
- Linux Mint
- Kali Linux
- Pop!_OS

Unlike `apt`, `dpkg` **does not automatically download packages or resolve dependencies**. It only works with packages already available on your system.

Syntax

```
dpkg [options] package
```

or

```
dpkg [options] package.deb
```

Common Options

Option	Description
`-i`	Install a package
`-r`	Remove a package (keep configuration files)
`-P`	Purge package (remove package and configuration files)
`-l`	List installed packages
`-L`	List files installed by a package
`-S`	Find which package owns a file
`-s`	Show package status
`--configure`	Configure unpacked packages
`--get-selections`	Show package selections
`--clear-selections`	Clear package selections

Examples

1. Install a `.deb` package

```
sudo dpkg -i google-chrome.deb
```

Example:

```
sudo dpkg -i vscode.deb
```

Output:

```
Selecting previously unselected package code.
Unpacking code...
Setting up code...
```

---

2. Install multiple packages

```
sudo dpkg -i *.deb
```

---

3. Remove a package

Keeps configuration files.

```
sudo dpkg -r nginx
```

---

4. Purge a package

Removes package and its configuration files.

```
sudo dpkg -P nginx
```

5. List installed packages

```
dpkg -l
```

Example output:

```
ii  bash
ii  coreutils
ii  openssh-server
ii  vim
```

---

6. Search for a package

```
dpkg -l | grep nginx
```

Output:

```
ii nginx
```

---

7. Show package information

```
dpkg -s nginx
```

Example:

```
Package: nginx
Status: install ok installed
Version: 1.24.0
Architecture: amd64
```

---

8. List files installed by a package

```
dpkg -L nginx
```

Example output:

```
/usr/sbin/nginx
/etc/nginx/
/etc/nginx/nginx.conf
```

9. Find which package owns a file

```
dpkg -S /usr/bin/ssh
```

Output:

```
openssh-client: /usr/bin/ssh
```

---

10. Reconfigure a package

```
sudo dpkg --configure package-name
```

Example:

```
sudo dpkg --configure openssh-server
```

---

11. Configure all unpacked packages

Useful after an interrupted installation.

```
sudo dpkg --configure -a
```

---

12. Check whether a package is installed

```
dpkg -s curl
```

or

```
dpkg -l curl
```

---

13. Show package architecture

```
dpkg -s libc6
```

Output:

```
Architecture: amd64
```

14. Show all installed packages

```
dpkg --get-selections
```

Example:

```
bash                    install
curl                    install
vim                     install
```

Package Status Codes (`dpkg -l`)

When you run:
`dpkg -l`

the first two letters indicate the package's status.

Code	Meaning
`ii`	Installed
`rc`	Removed, configuration files remain
`un`	Unknown/Not installed
`hi`	Half-installed
`pn`	Purged

Example:

```
ii  bash
ii  vim
rc  apache2
```

Fix Broken Installations

If installing a package fails because dependencies are missing:

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

You may see an error like:

```
dependency problems prevent configuration
```

To fix it:

```
sudo apt --fix-broken install
```

This installs any missing dependencies and completes the installation.

Difference Between `dpkg` and `apt`

Feature	`dpkg`	`apt`
Installs `.deb` files		
Downloads packages from repositories	✘	
Resolves dependencies automatically	✘	
Removes packages		
Updates package lists	✘	(`apt update`)
Upgrades installed packages	✘	(`apt upgrade`)

When to Use `dpkg`

Use `dpkg` when you:

- Have a local `.deb` file to install.
- Need to inspect installed packages.
- Want to find which package owns a file.
- Need to manually configure or repair package installations.
- Want detailed package information.

Use `apt` when you:

- Install software from repositories.
- Need automatic dependency resolution.
- Update or upgrade your system.
- Search for available packages.

Summary

- **Command:** `dpkg`
- **Purpose:** Manage Debian (`.deb`) packages locally.
- **Works on:** Debian, Ubuntu, Linux Mint, Kali Linux, Pop!_OS, and other Debian-based distributions.
- **Does not resolve dependencies automatically**—use `apt` for that.
- Common options:
  - `-i` → Install
  - `-r` → Remove
  - `-P` → Purge
  - `-l` → List installed packages
  - `-L` → List files in a package
  - `-S` → Find the package owning a file
  - `-s` → Show package status
  - `--configure -a` → Configure all unpacked packages (often used to recover from interrupted installations)

3.dnf

`dnf` (**Dandified YUM**) is the default **package manager** for Red Hat-based Linux distributions such as:

- Fedora
- Red Hat Enterprise Linux (RHEL) 8 and later
- CentOS Stream
- Rocky Linux
- AlmaLinux

It is used to **install, update, remove, search, and manage software packages** from software repositories.

Think of `dnf` as the Linux equivalent of an app store command-line tool.

Syntax

```
dnf [options] <command> [package]
```

Common Commands

1. Install a package

```
sudo dnf install httpd
```

Installs the Apache web server.

---

2. Remove a package

```
sudo dnf remove httpd
```

Removes the installed package.

3. Update all packages

```
sudo dnf update
```

or

```
sudo dnf upgrade
```

Updates all installed packages to their latest versions.

---

4. Update a specific package

```
sudo dnf update vim
```

Updates only the `vim` package.

---

5. Search for a package

Example output:

```
nginx.x86_64
nginx-all-modules
nginx-filesystem
```

---

6. Display package information

```
dnf info nginx
```

Shows:

- Version
- Repository
- Size
- Description
- Architecture

7. List installed packages

```
dnf list installed
```

---

8. List available packages

```
dnf list available
```

---

9. Check for updates

```
dnf check-update
```

Lists packages that have newer versions available.

---

10. Reinstall a package

```
sudo dnf reinstall bash
```

Useful if package files become corrupted.

---

11. Clean downloaded package cache

```
sudo dnf clean all
```

Removes:

- Cached packages
- Metadata
- Repository cache

12. Show package dependencies

```
dnf deplist httpd
```

Displays packages required by `httpd`.

---

13. Find which package provides a file

```
dnf provides /usr/bin/python3
```

or

```
dnf provides "*/python3"
```

Very useful when you know a filename but not the package that contains it.

---

14. Show repository list

```
dnf repolist
```

Example:

```
repo id              repo name
appstream            AppStream
baseos               BaseOS
extras               Extras
```

---

15. Enable a repository

```
sudo dnf config-manager --set-enabled powertools
```

---

16. Disable a repository

```
sudo dnf config-manager --set-disabled powertools
```

17. View package history

```
dnf history
```

Shows:

- Installations
- Updates
- Removals
- Rollbacks

---

18. Undo a transaction

```
sudo dnf history undo 15
```

Reverts transaction ID 15.

---

19. Autoremove unused packages

```
sudo dnf autoremove
```

Removes packages that were installed as dependencies but are no longer needed.

---

20. List package groups

```
dnf group list
```

Example groups:

- Development Tools
- Server
- KDE Plasma
- GNOME Desktop

21. Install a package group

```
sudo dnf group install "Development Tools"
```

Installs a collection of related packages.

Useful Options

Option	Description
`-y`	Automatically answer "yes" to prompts
`-q`	Quiet mode
`-v`	Verbose output
`--refresh`	Refresh repository metadata before running
`--best`	Use the best available package version
`--allowerasing`	Allow removal of conflicting packages if needed

Example:

```
sudo dnf install nginx -y
```

Common Examples

Install Git:

```
sudo dnf install git
```

Update the entire system:

```
sudo dnf update
```

Search for Docker:

```
dnf search docker
```

Remove Docker:

```
sudo dnf remove docker
```

Check package information:

```
dnf info git
```

Find which package contains `ifconfig`:

```
dnf provides */ifconfig
```

Exit Status

- **0** – Command completed successfully.
- **1** – General error (for example, invalid arguments or package not found).
- **100** – Updates are available (commonly returned by `dnf check-update`).

When to Use `dnf`

Use `dnf` when you need to:

- Install new software.
- Update installed packages.
- Remove software.
- Search for available packages.
- Manage software repositories.
- View package information.
- Resolve package dependencies.
- Roll back package transactions.

`yum` (vs) `dnf`

Feature	`yum`	`dnf`
Default in older RHEL/CentOS		
Default in Fedora/RHEL 8+		
Faster dependency resolution		
Better dependency solver		
Lower memory usage		
Transaction history	Basic	Improved
Active development	Limited	

Most `yum` commands continue to work on modern systems because `yum` is often provided as a compatibility wrapper around `dnf`.

Summary

- **Command:** `dnf`
- **Full Form:** Dandified YUM
- **Purpose:** Manage software packages on Red Hat-based Linux distributions.
- **Requires root privileges** (`sudo`) for installing, updating, or removing packages.
- **Key operations:** Install, update, remove, search, query, manage repositories, inspect dependencies, and review or undo package transactions.

4.rpm

The `rpm` command is a package management utility used in **Red Hat-based Linux distributions** such as:

- Red Hat Enterprise Linux (RHEL)
- CentOS
- Rocky Linux
- AlmaLinux
- Fedora
- Oracle Linux

It is used to **install, upgrade, remove, verify, and query** software packages in the **RPM (.rpm)** format.

> **Note:** `rpm` manages individual package files. It **does not automatically resolve dependencies**. For dependency management, use `dnf` (Fedora, RHEL 8+) or `yum` (older RHEL/CentOS).

Common Options

Option	Description
`-i`	Install a package
`-U`	Upgrade a package
`-F`	Freshen (upgrade only if already installed)
`-e`	Erase (remove) a package
`-q`	Query installed packages
`-V`	Verify installed package
`-K`	Verify RPM package signature/checksum
`-h`	Print hash marks during installation
`-v`	Verbose output

Installing a Package

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

Example

```
rpm -ivh httpd-2.4.57.rpm
```

Options used:

- `-i` → Install
- `-v` → Verbose
- `-h` → Display installation progress using #

Example output

```
Preparing... ########################
Installing... ########################
```

Upgrading a Package

```
rpm -Uvh package.rpm
```

Example

```
rpm -Uvh vim-9.0.rpm
```

If an older version exists:

- Old version removed
- New version installed

Freshen a Package

Only upgrades if already installed.

```
rpm -Fvh package.rpm
```

If package isn't installed

```
package not installed
```

Remove a Package

```
rpm -e package_name
```

Example

```
rpm -e httpd
```

Removes Apache.

List Installed Packages

```
rpm -qa
```

Example

```
bash-5.1
kernel-5.14
vim-9.0
openssl-3.0
```

`-q` → Query

`-a` → All packages

Search for a Package

```
rpm -qa | grep vim
```

Output

```
vim-enhanced
vim-common
```

Check Whether a Package is Installed

```
rpm -q httpd
```

Output:

```
httpd-2.4.57
```

If absent

```
package httpd is not installed
```

Show Package Information

```
rpm -qi httpd
```

Example

```
Name        : httpd
Version     : 2.4.57
Release     : 1
Architecture: x86_64
Install Date: Mon Jul 27
Summary     : Apache HTTP Server
```

List Files Installed by a Package

```
rpm -ql httpd
```

Example

```
/etc/httpd
/usr/sbin/httpd
/var/www/html
```

Find Which Package Owns a File

```
rpm -qf /usr/bin/vim
```

Output

```
vim-enhanced-9.0
```

Very useful when you want to know which package installed a particular file.

Verify an Installed Package

```
rpm -V httpd
```

Checks whether package files have been modified.

No output means

```
Package is unchanged.
```

Verify an RPM File Before Installing

```
rpm -K package.rpm
```

Example

```
rpm -K httpd.rpm
```

Output

```
httpd.rpm: digests OK
```

Query an RPM File Without Installing

```
rpm -qpi package.rpm
```

Shows package information.

```
rpm -qpl package.rpm
```

Lists files contained in the RPM.

Install Without Dependency Checks (Not Recommended)

```
rpm -ivh --nodeps package.rpm
```

Use only when you understand the consequences, as missing dependencies may cause the software to fail.

Force Installation

```
rpm -ivh --force package.rpm
```

Overwrites an existing package if necessary.

Common Examples

Install

```
rpm -ivh nginx.rpm
```

Upgrade

```
rpm -Uvh nginx.rpm
```

Remove

```
rpm -e nginx
```

List installed packages

```
rpm -qa
```

Package info

```
rpm -qi nginx
```

Files in package

```
rpm -ql nginx
```

Owner of a file

```
rpm -qf /usr/sbin/nginx
```

Verify package

```
rpm -V nginx
```

Exit Status

Exit Code	Meaning
`0`	Command completed successfully
`1`	General error (e.g., package not found, install failed)
`2`	Invalid command-line options or usage error

Advantages

- Fast and reliable package installation.
- Verifies package integrity and digital signatures.
- Provides detailed package information.
- Can verify installed files for tampering.
- Standard package manager for Red Hat-based distributions.

Disadvantages

- Does **not** automatically resolve dependencies.
- Installing packages manually can become cumbersome when many dependencies are required.
- Higher-level tools like `dnf` or `yum` are generally preferred for routine package management.

Summary

Command	Purpose
`rpm -ivh package.rpm`	Install a package
`rpm -Uvh package.rpm`	Upgrade a package
`rpm -Fvh package.rpm`	Freshen an installed package
`rpm -e package`	Remove a package
`rpm -qa`	List all installed packages
`rpm -qi package`	Show package information
`rpm -ql package`	List files installed by a package
`rpm -qf file`	Find which package owns a file
`rpm -V package`	Verify installed package integrity
`rpm -K package.rpm`	Verify an RPM file before installation

Note: On modern Red Hat-based systems, use `dnf` (or `yum` on older releases) for most package management tasks because it automatically downloads required dependencies, while `rpm` is best suited for working directly with `.rpm` package files and performing low-level package queries or verification.

5. pacman

`pacman` (Package Manager) is the **default package management tool for Arch Linux and Arch-based distributions** (such as Manjaro, EndeavourOS, Garuda Linux, etc.). It is used to **install, update, remove, search, and manage software packages**.

> **Note:** `pacman` is **not available on Ubuntu, Debian, Fedora, or RHEL**. It is specific to Arch-based systems.

Syntax

```
pacman [options] [package_name]
```

Common Operations

1. Update package database

```
sudo pacman -Sy
```

Downloads the latest package database from repositories.

Example:

```
sudo pacman -Sy
```

---

2. Upgrade all installed packages

```
sudo pacman -Syu
```

This is the most commonly used command.

Example:

```
sudo pacman -Syu
```

What it does:

- Synchronizes package databases (`-Sy`)
- Upgrades all installed packages (`-u`)

---

3. Install a package

```
sudo pacman -S package_name
```

Example:

```
sudo pacman -S firefox
```

Installs Firefox.

---

4. Install multiple packages

```
sudo pacman -S vim git wget
```

Installs all three packages.

---

5. Remove a package

```
sudo pacman -R package_name
```

Example:

```
sudo pacman -R firefox
```

Removes Firefox.

6. Remove package with unused dependencies

```
sudo pacman -Rs package_name
```

Example:

```
sudo pacman -Rs firefox
```

Removes Firefox and dependencies that are no longer needed.

---

7. Remove package completely

```
sudo pacman -Rns package_name
```

Example:

```
sudo pacman -Rns firefox
```

Removes:

- Package
- Configuration files
- Unused dependencies

---

8. Search for a package

```
pacman -Ss package_name
```

Example:

```
pacman -Ss vlc
```

Displays matching packages.

9. Search installed packages

```
pacman -Qs package_name
```

Example:

```
pacman -Qs firefox
```

Shows installed packages matching the search.

---

10. Display package information

```
pacman -Si package_name
```

Example:

```
pacman -Si firefox
```

Shows:

- Version
- Repository
- Size
- Dependencies
- Description

11. Show installed package information

```
pacman -Qi package_name
```

Example:

```
pacman -Qi firefox
```

Shows:

- Installation date
- Install reason
- Version
- Dependencies
- Package size

12. List all installed packages

```
pacman -Q
```

Example:

```
pacman -Q
```

Lists every installed package.

---

13. Check if a package is installed

```
pacman -Q package_name
```

Example:

```
pacman -Q git
```

Output:

```
git 2.51.0-1
```

---

14. List files installed by a package

```
pacman -Ql package_name
```

Example:

```
pacman -Ql git
```

Displays all files belonging to the package.

---

15. Find which package owns a file

```
pacman -Qo /path/to/file
```

Example:

```
pacman -Qo /usr/bin/git
```

Output:

```
/usr/bin/git is owned by git
```

16. Download packages without installing

```
sudo pacman -Sw package_name
```

Example:

```
sudo pacman -Sw firefox
```

Downloads the package into the package cache.

---

17. Clean package cache

Remove cached packages that are no longer installed:

```
sudo pacman -Sc
```

Remove **all** cached packages:

```
sudo pacman -Scc
```

Use `-Scc` with caution, as it removes all cached package files.

---

18. List orphaned packages

```
pacman -Qdt
```

Lists installed packages that are no longer required by any other package.

Remove them:

```
sudo pacman -Rns $(pacman -Qdtq)
```

---

19. List available repositories

```
pacman -Sl
```

Shows all packages available in enabled repositories.

Frequently Used Options

Option	Description
`-S`	Install packages from repositories
`-Sy`	Refresh package database
`-Syu`	Refresh database and upgrade the system
`-R`	Remove a package
`-Rs`	Remove package and unneeded dependencies
`-Rns`	Remove package, configuration files, and unneeded dependencies
`-Q`	Query installed packages
`-Qi`	Show installed package information
`-Si`	Show repository package information
`-Ss`	Search repositories
`-Qs`	Search installed packages
`-Ql`	List files owned by a package
`-Qo`	Find which package owns a file
`-Sw`	Download packages only
`-Sc`	Clean old package cache
`-Scc`	Remove all cached packages

Examples

Install Git:

```
sudo pacman -S git
```

Update the entire system:

```
sudo pacman -Syu
```

Search for VLC:

```
pacman -Ss vlc
```

Remove VLC:

```
sudo pacman -R vlc
```

Show installed package details:

```
pacman -Qi git
```

Find the owner of a file:

```
pacman -Qo /usr/bin/python
```

When to Use `pacman`

Use `pacman` when you need to:

- Install software packages
- Update your Arch Linux system
- Remove installed software
- Search for available packages
- Query installed packages
- View package information
- Manage dependencies and package caches

`pacman` vs Other Linux Package Managers

Distribution	Package Manager	Example Install Command
Arch Linux	`pacman`	`sudo pacman -S nginx`
Debian/Ubuntu	`apt`	`sudo apt install nginx`
Fedora	`dnf`	`sudo dnf install nginx`
RHEL/CentOS	`dnf` (or older `yum`)	`sudo dnf install nginx`
openSUSE	`zypper`	`sudo zypper install nginx`

Summary: `pacman` is the fast and efficient package manager used on Arch Linux and its derivatives. It provides a unified interface for installing, updating, removing, querying, and maintaining software packages and their dependencies.

6.zypper

`zypper` is the **command-line package manager** used in **openSUSE** and **SUSE Linux Enterprise (SLE/SLES)**. It allows you to install, update, remove, search, and manage software packages from repositories.
It works with **RPM packages** and uses the **libzypp** library for dependency resolution.

Syntax

```
zypper [options] <command> [arguments]
```

Common `zypper` Commands

Command	Description
`zypper refresh`	Refresh repository metadata
`zypper install package`	Install a package
`zypper remove package`	Remove a package
`zypper update`	Update installed packages
`zypper search package`	Search for a package
`zypper info package`	Display package information
`zypper list-updates`	Show available updates
`zypper repos`	List configured repositories
`zypper addrepo URL name`	Add a repository
`zypper removerepo name`	Remove a repository
`zypper clean`	Clean cached package data

Examples

1. Refresh Repository Information

```
sudo zypper refresh
```

or

```
sudo zypper ref
```

Downloads the latest package information from configured repositories.

---

2. Install a Package

```
sudo zypper install vim
```

or

```
sudo zypper in vim
```

Installs the **Vim** editor.

---

3. Remove a Package

```
sudo zypper remove vim
```

or

```
sudo zypper rm vim
```

Removes the package.

---

4. Update All Installed Packages

```
sudo zypper update
```

or

```
sudo zypper up
```

Updates all installed packages to the latest available versions.

5. Upgrade Distribution

```
sudo zypper dist-upgrade
```

or

```
sudo zypper dup
```

Performs a distribution upgrade by resolving package replacements, removals, and dependency changes.

---

6. Search for a Package

```
zypper search firefox
```

or

```
zypper se firefox
```

Searches repositories for packages related to **Firefox**.

---

7. Show Package Information

```
zypper info firefox
```

or

```
zypper if firefox
```

Displays details such as:

- Version
- Repository
- Architecture
- Summary
- Description

8. List Installed Packages

```
zypper search --installed-only
```

or

```
zypper se -i
```

Shows installed packages.

---

9. List Available Updates

```
zypper list-updates
```

or

```
zypper lu
```

Lists packages with updates available.

---

10. List Repositories

```
zypper repos
```

or

```
zypper lr
```

Displays configured software repositories.

Example output:

```
# | Alias | Name | Enabled | GPG Check
-----------------------------------------
1 | repo-oss | Main Repository | Yes | Yes
2 | repo-update | Update Repository | Yes | Yes
```

11. Add a Repository

```
sudo zypper addrepo https://download.example.com/repo myrepo
```

or

```
sudo zypper ar https://download.example.com/repo myrepo
```

Adds a new software repository.

---

12. Remove a Repository

```
sudo zypper removerepo myrepo
```

or

```
sudo zypper rr myrepo
```

Deletes a configured repository.

---

13. Clean Package Cache

```
sudo zypper clean
```

or

```
sudo zypper cc
```

Removes cached package files and metadata.

---

14. Verify Dependencies

```
sudo zypper verify
```

Checks installed packages for broken dependencies and attempts to fix them.

15. Download Packages Without Installing

```
zypper install --download-only nginx
```

Downloads the package but does not install it.

---

16. Install from a Local RPM File

```
sudo zypper install ./mypackage.rpm
```

Installs a local RPM package and resolves any dependencies automatically.

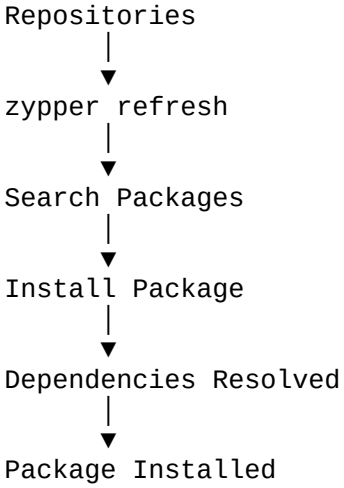
Useful Options

Option	Description
`-y`	Automatically answer "yes" to prompts
`--dry-run`	Simulate the operation without making changes
`--no-recommends`	Install only required dependencies
`--force`	Force installation or reinstallation
`--download-only`	Download packages without installing
`--details`	Show detailed output
`--quiet`	Reduce output verbosity

Example:

```
sudo zypper install -y nginx
```

Package Management Workflow



Comparison with Other Linux Package Managers

Distribution	Package Manager	Package Format
openSUSE	`zypper`	RPM
SLES	`zypper`	RPM
Fedora	`dnf`	RPM
RHEL	`dnf` / `yum`	RPM
Debian	`apt`	DEB
Ubuntu	`apt`	DEB
Arch Linux	`pacman`	PKG

Advantages of `zypper`

- Fast dependency resolution
- Supports RPM packages
- Automatic dependency installation
- Repository management
- Distribution upgrades (`dup`)
- Package verification
- Pattern and patch management
- Script-friendly command-line interface

Summary

- **Purpose:** Manage software packages and repositories on openSUSE and SUSE Linux Enterprise.
- **Package Format:** RPM
- **Common Tasks:** Install, remove, update, search, verify, and manage repositories.
- **Typical Usage:** `sudo zypper install <package>` to install software and `sudo zypper update` to keep the system up to date.