ADMIN:DOCS

openSUSE: the Zypper package manager guide for beginners

Understand package management on openSUSE: the libzypp SAT solver, Leap 16 vs. Tumbleweed, patterns, Packman prioritization, Snapper rollbacks and OPI in practice.

Category: Linux beginners **Published:** 23.10.2024 **Last reviewed:** 08.09.2026

Among the major Linux distributions, openSUSE occupies a quite particular place. Founded in 1992 in Nuremberg in Franconia, the distribution with the green chameleon *Geeko* looks back on more than thirty years of development. What makes openSUSE attractive for beginners and professionals alike is the successful combination of German engineering, uncompromising enterprise stability from its kinship with *SUSE Linux Enterprise* (SLE), and a highly modern package-management system: **Zypper**.

Anyone switching to openSUSE from other distributions notices the difference immediately:

Zypper is remarkably precise, speaks plainly about dependencies and behaves, in the best sense, predictably. Behind the `zypper` command, the C++ library `libzypp` is a genuine pioneering achievement: openSUSE was the first Linux system in the world to use a mathematical **SAT solver** to resolve package conflicts — a principle of propositional logic that was later adopted even by other package managers.

The most spectacular feature of openSUSE, however, is the built-in safety net: Thanks to the deep symbiosis of Zypper, the `Btrfs` filesystem and the snapshot tool `Snapper`, the system automatically creates backups of the entire operating system on every package installation. If a driver or a new software version ever takes the desktop down, you simply pick the state from ten minutes ago in the boot menu at startup and continue working as if nothing had happened.

Source: https://admindocs.de/en/linux-beginners/opensuse-zypper-package-manager-guide

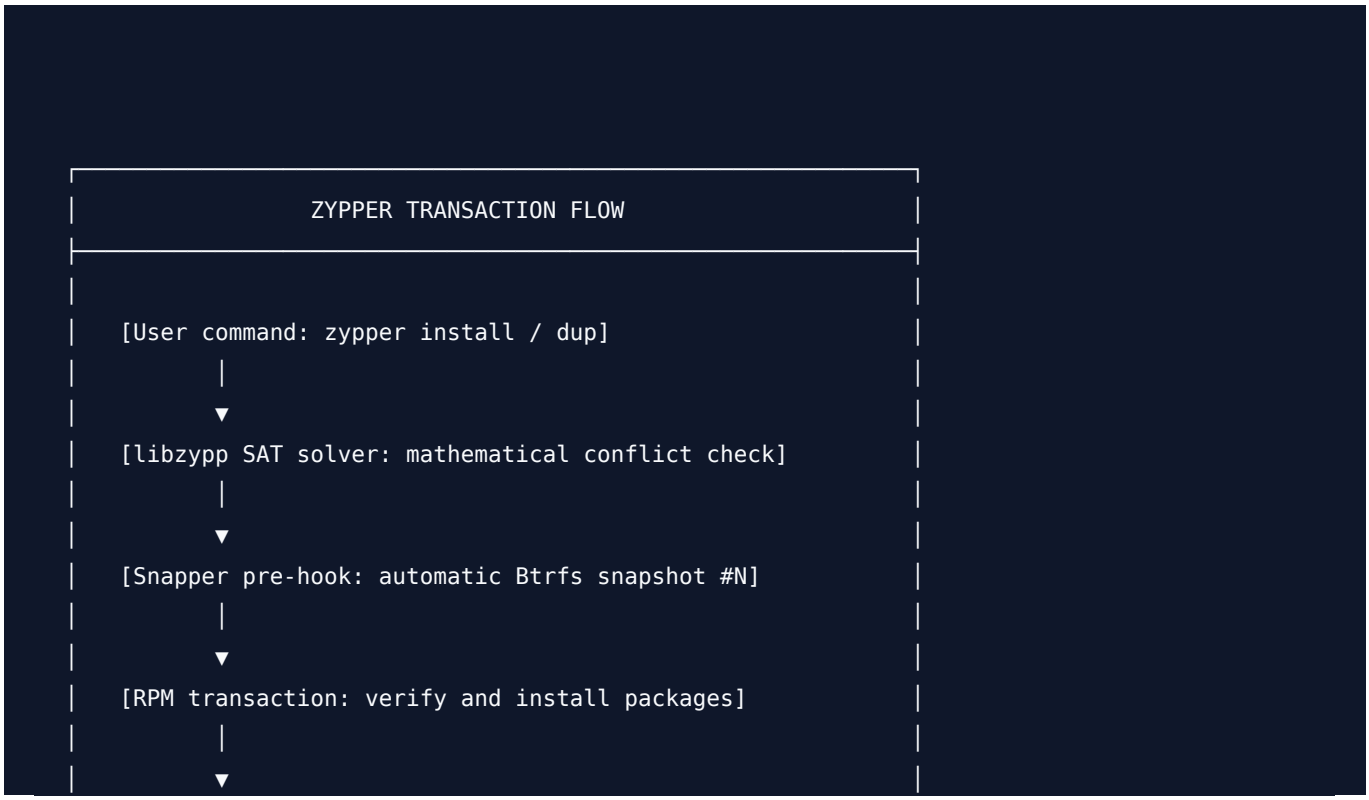
You will learn the differences between the stable version Leap 16.0 and the innovative rolling release Tumbleweed, master installation via patterns, add multimedia codecs with little friction using OPI, and use Snapper snapshots as your personal fully comprehensive insurance.

> **Basic terms at a glance:** `Zypper` works with **RPM packages** (`.rpm`), organizes software in **repositories** (software sources) and groups functional software bundles (such as complete desktop environments or developer tools) into so-called **patterns**.

Architecture and fundamentals: how Zypper and libzypp work

`Zypper` itself is the comfortable command-line user interface. The actual brain in the background is the performant C++ library `libzypp`.

When you install software or update the system, a multi-stage, fully automatic process runs:



```
   │   [Snapper post-hook: Btrfs snapshot #N+1 with diff]        │
   │                                                             │
   └─────────────────────────────────────────────────────────────┘
```

1. Mathematical precision: the SAT solver

Conventional package managers used to rely on simple heuristics to guess dependencies, which regularly led to dead ends on tangled package conflicts. `libzypp` instead translates dependencies, version requirements and system architectures into a mathematical satisfiability problem (*Boolean Satisfiability Problem*, SAT for short).

The solver computes exactly the intersection that keeps the system stable. If there are two contradictory solutions (for example because a library is provided by two competing versions), Zypper never guesses arbitrarily, but asks you interactively with clear options.

2. The Snapper shield

Before a single new file is written to disk, the `zypp-plugin-snapper` plugin triggers a so-called *pre-snapshot* on the Btrfs filesystem. As soon as all RPM packages have been unpacked, configured and services restarted, the *post-snapshot* follows. The Snapper database records exactly which files changed between the two points in time. If the installation fails, the way back is secured with a keypress.

The openSUSE family: Leap, Tumbleweed and Slowroll

Before you work with Zypper, you must know which edition of openSUSE you have in front of you. System care differs fundamentally here:

- **openSUSE Leap 16.0:** The reliable point release for anyone seeking absolute stability. Leap 16.0 is based on the modern enterprise platform **SUSE Linux Enterprise 16 (SLE 16)** and the modular **SUSE Linux Framework One (SLFO)**. The software set stays stable for years and receives continuous security updates.

- **openSUSE Tumbleweed:** The highly modern, continuous **rolling release**. There are no version numbers here; you receive day-current Linux kernels, drivers and desktop environments. So that Tumbleweed stays rock-solid despite the pace, every software snapshot runs through fully automatic tests in a huge virtual test farm named **openQA** before release.
- **openSUSE Slowroll:** The golden middle for users who find Tumbleweed too fast-moving and Leap too conservative. Slowroll takes packages from Tumbleweed, but rolls larger updates out in calm, monthly proven waves, while critical security patches arrive immediately.

Package sources and repository architecture

Like all modern distributions, openSUSE fetches its software from structured online archives. By default four main sources are configured on the system:

- **OSS (Open Source Software):** The foundation with all free, open-source programs and libraries.
- **Non-OSS:** Proprietary software, hardware drivers (e.g. for graphics cards) and programs with non-free licenses.
- **Update (OSS and Non-OSS):** Continuously maintained security patches and bug fixes for the running system.
- **Backports:** More current software packages provided specifically for the Leap base.

Configuration in /etc/zypp/repos.d/

Each software source is stored as a structured `.repo` file in `/etc/zypp/repos.d/`:

```
[repo-oss]
name=openSUSE-Leap-OSS
enabled=1
```

```
autorefresh=1
baseurl=http://download.opensuse.org/distribution/leap/16.0/repo/oss/
type=rpm-md
gpgcheck=1
gpgkey=http://download.opensuse.org/distribution/leap/16.0/repo/oss/repodata/repomd.xml.ke
y
keeppackages=0
priority=99
```

- `enabled=1`: Determines whether the repository is used for queries and installations.
- `autorefresh=1`: Zypper automatically refreshes the server metadata in the background as soon as it is stale.
- `priority=99`: The priority of the source on a scale from 1 (highest) to 99 (default).

Managing repositories on the console

With the commands `repos` (short form `lr`), `addrepo` (`ar`), `modifyrepo` (`mr`) and `removerepo` (`rr`) you steer your sources flexibly:

□□ Practical example:

List, add and maintain repositories:

```
# Show all configured repositories with priority (-p) and URLs (-u)
zypper lr -d -p

# Add a new repository and enable autorefresh immediately (-f)
sudo zypper ar -f https://download.opensuse.org/repositories/benchmark/16.0/ benchmark

# Temporarily disable a repository (-d)
sudo zypper mr -d benchmark

# Enable a repository again (-e)
```

```
sudo zypper mr -e benchmark

# Remove a repository that is no longer needed without residue
sudo zypper rr benchmark
```

Package management in daily work: search, install and clean

Daily work with Zypper is intuitive. Many commands have practical two-letter abbreviations, which saves administrators a lot of typing.

1. Finding packages and inspecting metadata

Before you install software you want to know the exact package name and which source it comes from:

□□ Practical example:

Search the catalog and check the package insert:

```
# Search package names for a term (short form: zypper se)
zypper search firefox

# Also search descriptions and summaries
zypper search -d "web browser"

# Filter already installed packages
zypper search -i nginx

# Show full package details (version, architecture, license, source)
zypper info firefox
```

2. Installing software

Installation uses the `install` command (short form: `in`):

□□ Practical example:

Install packages individually, in a bundle or from local RPM files:

```
# Install a single package
sudo zypper install htop

# Install several packages in a shared transaction
sudo zypper in git vim tmux curl

# Specify a particular package version
sudo zypper in "firefox = 147.0"

# Install a manually downloaded local RPM file
sudo zypper in ./my-program-1.0.0.rpm
```

During installation Zypper presents a precise preview of the transaction:

```
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following 3 NEW packages are going to be installed:
  firefox firefox-lang-en mozilla-nss

3 new packages to install.
Overall download size: 85.6 MiB. Already cached: 0 B.
After the operation, additional 216.8 MiB will be used.
Continue? [y/n/v/d/? shows all options] (y):
```

Interactive keys are available here:

- `y` (Yes): Confirm the transaction and run it.
- `n` (No): Abort without changes.
- `v` (Verbose): Shows detailed version and architecture jumps.
- `d` (Download only): Downloads the RPM files into the cache only, without installing them.

> **Automation in scripts:** When you install software in shell scripts and want prompts answered automatically with yes, use the `-n` switch (*non-interactive*): `sudo zypper -n in htop`.

3. The openSUSE peculiarity: installing whole worlds with patterns

openSUSE bundles related software not only in metapackages, but in so-called **patterns**. A pattern is a first-class citizen in Zypper and groups complete application areas:

Practical example:

Working with patterns:

```
# List all patterns available and installed on the system
zypper patterns

# Install a complete desktop environment with one command
sudo zypper in -t pattern kde_plasma

# Set up a complete web-server stack
sudo zypper in -t pattern lamp_server

# Set up a complete C/C++ development environment
sudo zypper in -t pattern devel_basis
```

4. Uninstalling software and keeping the system clean

When a program is no longer needed, you remove it with `remove` (short form: `rm`). To avoid orphaned libraries you should always use the `--clean-deps` switch (short form: `-u`):

□□ Practical example:

Uninstall programs and clean orphans:

```
# Remove a program and delete unused dependencies at the same time
sudo zypper remove -u firefox

# Find all orphaned packages no longer needed by any program
zypper packages --unneeded

# Wipe unused packages from the system
sudo zypper rm -u $(zypper --quiet packages --unneeded | awk -F '|' 'NR>2 {print $3}')
```

The iron update rule: update vs. dist-upgrade (dup)

On openSUSE there are two fundamentally different update commands. Mixing these two up is the most common cause of problems for beginners:

```
┌──────────────────────────────────────────────────────┐
│              UPDATE STRATEGY BY DISTRIBUTION         │
├──────────────────────────────────────────────────────┤
│                                                      │
│   openSUSE Leap (stable point release):              │
│   ├── Ongoing updates:     sudo zypper update        │
│   └── Version jump:        sudo zypper dup --releasever=16  │
│                                                      │
```

```
│   openSUSE Tumbleweed (rolling release):                    │
│   └── Only path:           sudo zypper dup                  │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

The two commands compared directly

Command	Purpose	How it works and the rules
`sudo zypper update` (`up`)	**openSUSE Leap** in daily use	Updates packages within the existing version branch. Never deletes packages and never switches vendors.
`sudo zypper dist-upgrade` (`dup`)	**openSUSE Tumbleweed** (always!) and major-version upgrades on Leap	Aligns the entire system exactly to the latest distribution state. May remove, replace and switch vendors to resolve conflicts.

> ⚠ **Life-critical rule for Tumbleweed users:** If you use openSUSE Tumbleweed, **never** run `zypper up` ! Tumbleweed is a rolling release in which libraries are swapped or renamed daily. `zypper up` refuses to delete old packages, which puts the system into a torn, unstable state. On Tumbleweed the only correct command is: `sudo zypper dup` .

Targeted patch management on Leap

On openSUSE Leap the security team provides updates as official **patches** that bundle several security-relevant bug fixes into logical packages:

□□ Practical example:

Apply security patches in a targeted way:

```
# Inspect all pending security and stability patches
zypper patches

# Show a summary of all vulnerabilities
zypper patch-info

# Install only security-critical patches
sudo zypper patch --category security
```

After the update: checking running processes and services (zypper ps)

After a system update, new program files and libraries sit on disk. Background services already running (such as web servers, databases or desktop components) still use the old, already deleted file versions in RAM.

With `zypper ps` (short tabular view: `zypper ps -s`) Zypper quickly analyzes which processes should be restarted after an update:

□□ Practical example:

Find processes with stale libraries:

```
# Fast overview of all processes that still hold deleted files in RAM
zypper ps -s
```

If the output shows, for example, the `nginx` or `sshd` service, you simply restart the corresponding service with `sudo systemctl restart nginx`. If the update affects fundamental core components such as `systemd`, `glibc` or the Linux kernel, a regular reboot (`sudo reboot`) is advisable.

Third parties, priorities and the vendor-sticky policy

By default openSUSE follows a strict **vendor-sticky policy**: a package once installed from the official openSUSE repository is never automatically replaced by a package of the same name from a third-party source — even if the third-party source offers a higher version number. That protects the system against unnoticed foreign software.

The Packman repository for multimedia and codecs

Because of software patents, openSUSE does not ship proprietary audio and video codecs (such as H.264, H.265 or AAC) in the default set. For media playback (VLC, OBS Studio, browser streaming) you need the community repository **Packman**.

So that Packman packages may replace the standard packages, you set a higher priority for Packman (lower numeric value, e.g. `90` instead of the default `99`):

□□ Practical example:

Add Packman by hand and allow a vendor change:

```
# 1. Add and prioritize the Packman repository for the current version (priority 90)
sudo zypper ar -cfp 90 https://ftp.gwdg.de/pub/linux/misc/packman/suse/openSUSE_Leap_16.0/
packman

# 2. Switch existing multimedia packages to Packman in a targeted way
sudo zypper dup --from packman --allow-vendor-change
```

The modern shortcut: OPI (openSUSE Package Installer)

The manual path through repositories and vendor changes is tedious for beginners. In practice the community therefore uses the clever helper **OPI**:

□□ Practical example:

Install OPI and set up multimedia codecs with one command:

```
# Install OPI from the standard sources
sudo zypper in opi

# Fully automatic codec installation (sets up Packman and handles vendor changes)
opi codecs

# Interactively search any software from the openSUSE Build Service (OBS)
opi vscode
```

`opi` asks you interactively for the desired source, adds the matching repository, installs the software and then asks whether the repository should be stored permanently or cleaned up again after the transaction.

Locking packages with package locks

If you want to prevent a specific software version (for example a particular kernel or a database) from being updated during a `zypper dup`, you use locks:

□□ Practical example:

Manage package locks:

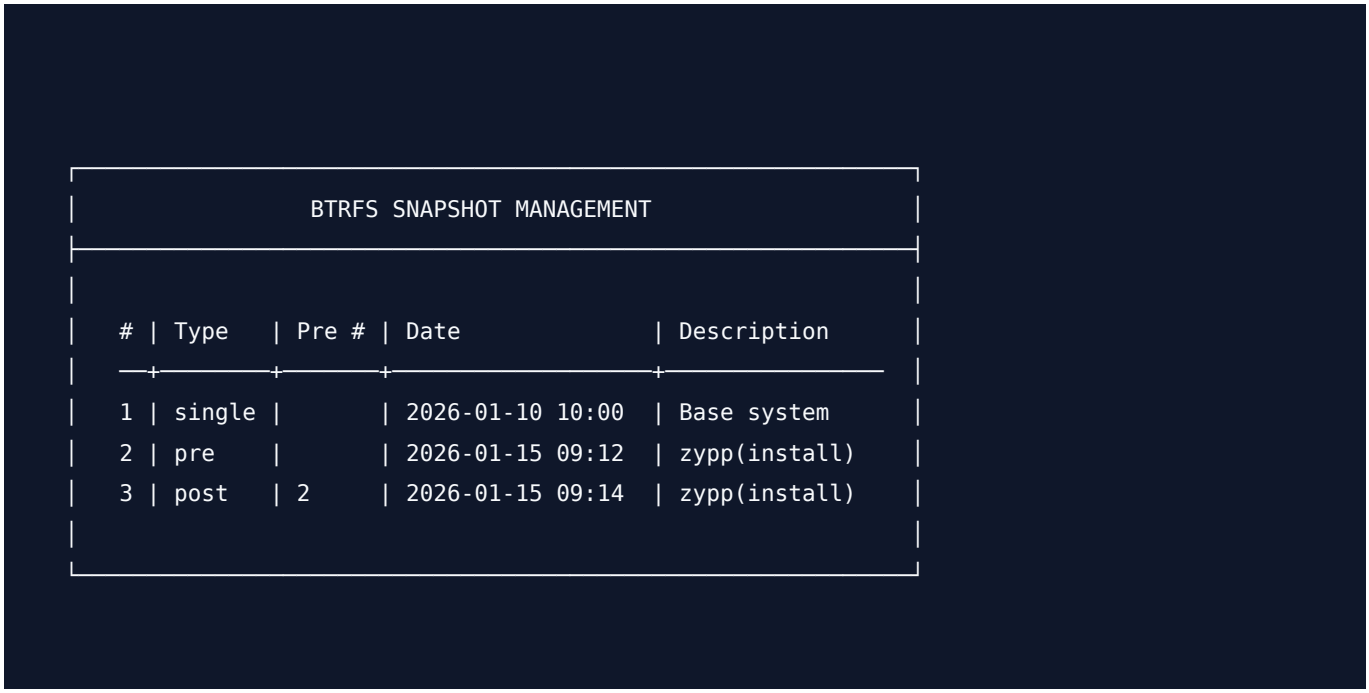
```
# Protect a package from all updates (short form: zypper al)
sudo zypper addlock kernel-default

# Show all currently active locks
zypper locks

# Lift the lock again (short form: zypper rl)
sudo zypper removelock kernel-default
```

The safety net: Snapper and automatic Btrfs rollbacks

The largest safety net on openSUSE is **Snapper**. On almost every Linux distribution an incomplete update or a broken graphics driver means lengthy tinkering in a rescue system. On openSUSE with the default Btrfs filesystem it is different.



Inspecting and comparing snapshots

You installed software and notice that the system shows unwanted behavior? With a few commands you analyze what changed:

□□ Practical example:

Manage snapshots and check differences:

```
# List all existing system snapshots
sudo snapper list

# Show changes between before-snapshot #2 and after-snapshot #3
```

```
sudo snapper status 2..3

# Compare exact file differences (close-up) of a configuration
sudo snapper diff 2..3 /etc/nginx/nginx.conf
```

Rolling the system back (rollback)

If there are serious problems, you roll the system back to an earlier snapshot:

Practical example:

Reset the system state:

```
# 1. Roll the system back to the stable previous state (snapshot #2)
sudo snapper rollback 2

# 2. Restart the machine
sudo reboot
```

After the reboot the machine is exactly in the state it had before the transaction. The previously broken state is not deleted, but remains as its own archive snapshot.

> **Last-resort rescue when the desktop no longer starts:** If the machine no longer boots to the graphical interface after an update, you do not need a live medium! In the **GRUB boot menu** simply choose *“Start bootloader from a read-only snapshot”*, select the desired snapshot and start the machine. Once you are logged in, run `sudo snapper rollback` in the terminal and reboot — done!

Tuning, store hygiene and troubleshooting

Zypper also needs a little administrative attention from time to time. Here are the most proven practical tips:

1. Parallel downloads and performance tuning in /etc/zypp/zypp.conf

A noticeable highlight of modern openSUSE systems is native support for **parallel package downloads** in `libzypp`. Instead of downloading RPM files one after another, Zypper pulls several packages over the wire at once, which massively speeds up large Tumbleweed upgrades.

In the central configuration file `/etc/zypp/zypp.conf` you can fine-tune the number of simultaneous connections:

□□ Practical example:

Optimize download speed:

```
# /etc/zypp/zypp.conf

# Allow up to 10 simultaneous connections for parallel downloads
download.max_concurrent_connections = 10

# Up to 5 parallel downloads per server mirror
download.max_concurrent_connections_per_server = 5

# Download all packages completely before the actual installation step
commit.downloadMode = DownloadInAdvance
```

2. Cleaning the cache and freeing disk space

Downloaded RPM files and metadata caches land under `/var/cache/zypp/`:

□□ Practical example:

Empty caches:

```
# Check disk space used by the Zypper cache
du -sh /var/cache/zypp/

# Empty all downloaded RPM packages and temporary metadata without residue
sudo zypper clean -a
```

3. Resolving blocked lock files

If the message `System management is locked by the application with pid...` appears:

□□ Practical example:

Identify the blocking process:

```
# Check who currently blocks Zypper (e.g. YaST, PackageKit or GNOME Software)
sudo fuser -v /var/run/zypp.pid

# Is an automatic update check running in the background?
# ──> Please wait a moment until the background service has finished!
```

4. Cache and GPG repair

If Zypper reports damaged metadata or expired repositories:

□□ Practical example:

Resynchronize repositories and re-import GPG keys:

```
# Discard the cache completely and force a rebuild
sudo zypper clean -a
sudo zypper refresh -f

# Automatically import new keys if needed
sudo zypper --gpg-auto-import-keys refresh
```

Command Reference (Cheatsheet)

Category	Command (long form)	Short form	Function and purpose
Search	`zypper search <term>`	`zypper se`	Searches the package catalog by name
Details	`zypper info <package>`	`zypper if`	Shows full package metadata
Install	`sudo zypper install <package>`	`sudo zypper in`	Installs software including dependencies
Patterns	`sudo zypper in -t pattern <name>`	`sudo zypper in -t pattern`	Installs functional software groups
Remove	`sudo zypper remove -u <package>`	`sudo zypper rm -u`	Deletes a package including orphaned dependencies
Update	`sudo zypper update`	`sudo zypper up`	Standard update on **openSUSE Leap**
Dist-upgrade	`sudo zypper dist-upgrade`	`sudo zypper dup`	Only update method on **Tumbleweed**
Service check	`zypper ps -s`	`zypper ps`	Shows processes that must restart after updates

Category	Command (long form)	Short form	Function and purpose
Patches	`sudo zypper patch --category security`	–	Applies security fixes on Leap in a targeted way
Source list	`zypper repos -d -p`	`zypper lr -d -p`	Lists repositories with priority and URLs
Add source	`sudo zypper addrepo -f <url> <alias>`	`sudo zypper ar -f`	Adds a new software source with autorefresh
Delete source	`sudo zypper removerepo <alias>`	`sudo zypper rr`	Removes a configured repository
Lock	`sudo zypper addlock <package>`	`sudo zypper al`	Protects a package from all system updates
Unlock	`sudo zypper removelock <package>`	`sudo zypper rl`	Frees a locked package again
Empty cache	`sudo zypper clean -a`	`sudo zypper cc -a`	Cleans local RPM and metadata caches
Rollback	`sudo snapper rollback <id>`	–	Resets the system to a Btrfs snapshot

Further Resources

Resource	Description	Type
Official openSUSE documentation	Full manuals for openSUSE Leap and Tumbleweed	Official documentation
openSUSE Zypper reference portal	Official wiki portal for all Zypper commands and options	Manual and reference
Snapper project documentation	Documentation of the Btrfs snapshot and rollback system	Project documentation
openSUSE Build Service (OBS)	Central community packaging and repository system	Community platform

Source: https://admindocs.de/en/linux-beginners/opensuse-zypper-package-manager-guide

Resource	Description	Type
Packman portal	Most important source for multimedia codecs and third-party software	Community portal

Conclusion

With Zypper, openSUSE has one of the most mature and reliable package managers in the entire Linux landscape. Through the combination of the mathematically precise SAT solver in `libzypp`, strict vendor protection and seamless Btrfs-Snapper integration, openSUSE offers a stability level that has few equals even on demanding rolling-release installations.

Anyone who has internalized the golden rule — `zypper up` on Leap, but exclusively `zypper dup` on Tumbleweed — and steers multimedia packages elegantly through the community tool `opi`, runs a highly modern, very fast operating system that is practically indestructible thanks to automatic rollbacks.

> **Practical tip for daily work:** When you face a tricky system operation or a large driver upgrade, create a manual safety point in advance with a one-liner: `sudo snapper create -d "Before driver install"`. If the system then does not match what you want, `sudo snapper rollback` and a quick reboot put you back on your starting point in seconds.

If you want to go beyond classic, stateful package managers and get to know a fully declarative, immutable operating system, the next guide introduces NixOS and the functional package manager Nix.

Source: https://admindocs.de/en/linux-beginners/opensuse-zypper-package-manager-guide