ADMIN:DOCS

Fedora: the DNF package manager guide for beginners

Understand package management on Fedora and RHEL from the ground up: from RPM fundamentals and DNF5 architecture through repository management to transaction rollbacks and operational practice.

Category: Linux beginners **Published:** 20.09.2024 **Last reviewed:** 08.09.2026

Anyone coming from Windows to Fedora Linux often looks in vain for familiar patterns at first: there are no install wizards with countless “Next” buttons, no unchecked `.exe` files from shady download portals, and no driver CDs. On Linux, software delivery works in a fundamentally different way — centralized, cryptographically signed and fully automatic.

The heart of this system on Fedora, Red Hat Enterprise Linux (RHEL), AlmaLinux and Rocky Linux is **DNF** (**Dandified YUM**), or **DNF5** on modern installations.

For beginners a package manager first looks like a command-line app store. DNF does far more: it watches a dense mesh of tens of thousands of software components, resolves highly complex dependencies with mathematical precision, protects the operating system against version conflicts and logs every transaction in such detail that you can undo faulty system changes as if with a time machine.

The range runs from the basic RPM package format through the modern C++ standard DNF5 and third-party sources such as RPM Fusion and Fedora COPR all the way to atomic transaction rollbacks on a running system.

Source: https://admindocs.de/en/linux-beginners/fedora-dnf-package-manager-guide

The three pillars: how RPM, repositories and DNF work together

To use DNF safely, it helps to look at the three core building blocks of software distribution on Fedora:



1. The RPM format: the data container

The `.rpm` format (*Red Hat Package Manager*) is the elementary package archive. An `.rpm` file contains:

- The actual program files, libraries and configuration files in compressed form.
- Metadata: name, version number, license, architecture (e.g. `x86_64` or `aarch64`) and file size.

- Control scripts: actions that must run before or after installation (e.g. creating a system user).
- A cryptographic signature (GPG) that ensures the package is unchanged and comes from the Fedora developers.
- A precise declaration of all dependencies (*“I need libssl.so.3 and glibc >= 2.38”*).

The underlying `rpm` tool can unpack and register such packages directly on disk, but it hits limits immediately in daily work:

> **Why rpm alone is not enough:** If a package requires three further libraries, the command `rpm -ivh program.rpm` simply aborts with an error. It does not know where to load the missing pieces from. Administrators used to call this historical phenomenon the dreaded *dependency hell*.

2. Repositories: curated software sources

A **repository** (software source) is a structured web server on which thousands of RPM packages that fit together are provided together with a machine-readable contents overview (the *metadata*). Fedora operates mirrored server networks worldwide (mirrors) that ensure downloads are always fast and resilient.

3. DNF: the intelligent navigator

This is where **DNF** takes the helm: when you request a program, DNF loads repository metadata, checks the dependencies of all involved components, computes the optimal installation plan, downloads all required RPMs, verifies their GPG keys and hands the packages to `rpm` in order for the actual installation.

The generation change: what sets DNF5 apart on Fedora

With Fedora 41 the step that administrators had been working toward for years was completed: **DNF5 has finally replaced the classic, Python-based predecessor DNF (DNF4).** On

modern systems the `dnf` command points seamlessly at the new binary `/usr/bin/dnf5`. This architecture is also the new foundation in the enterprise world: both CentOS Stream 10 and Red Hat Enterprise Linux 10 (RHEL 10) use this modern C++ stack by default.



The most important new features and differences for beginners at a glance:

- **Pure C++ performance:** DNF5 was rewritten from the ground up in modern C++ and uses the core library `libdnf5`. Dropping the Python runtime leads to up to four times faster start times and a drastically reduced memory footprint.
- **Cross-platform standard (enterprise-ready):** What was proven on Fedora forms the shared technological base on RHEL 10 and CentOS Stream 10. Knowledge learned here transfers seamlessly to modern enterprise servers.
- **Shared D-Bus daemon (`dnf5daemon`):** Desktop tools such as GNOME Software or Cockpit now use the same background service as the command line. Locked databases or

diverging update states are a thing of the past.

- **Separate history database:** DNF5 keeps its transaction history in its own SQLite database under `/var/lib/dnf5/history/`. Operations still run with old DNF4 no longer appear in the new history.
- **No more DeltaRPM:** Because modern CPUs and broadband links transfer data faster than historical delta diffs can be decompressed, DNF5 drops DeltaRPMs by default.

Repository architecture and source management

Fedora splits software sources into separate areas so that stability, security and legal requirements (open-source licenses) stay transparently apart.

Standard repositories on Fedora

Every freshly installed Fedora system has three preconfigured main sources:

- **fedora:** The base repository with the frozen software state at the official release tag of the respective Fedora version.
- **updates:** The continuously maintained source for bug fixes, security patches and new kernel versions.
- **updates-testing:** Pre-releases of packages for testers. Disabled by default so production systems are not destabilized.

Tuning the configuration in /etc/dnf/dnf.conf

The central control file for the package manager is `/etc/dnf/dnf.conf`. With a few adjustments everyday behavior can be improved noticeably:

□□ Practical example:

We open the configuration with administrator rights and adjust central parameters:

```
[main]
gpgcheck=True
installonly_limit=3
clean_requirements_on_remove=True
best=False
skip_if_unavailable=True
max_parallel_downloads=10
fastestmirror=True
```

Meaning of the most important directives:

- `max_parallel_downloads=10`: Downloads up to 10 RPM files at the same time. On fast broadband links this shortens system upgrades considerably.
- `installonly_limit=3`: Controls how many kernel versions remain on the system at once. In the boot menu (GRUB) you then always have two older, working kernels at hand if a kernel update causes problems.
- `clean_requirements_on_remove=True`: Ensures that when a program is removed, all dependencies that no other package still needs are uninstalled as well.
- `fastestmirror=True`: Measures server latency during metadata refresh and prefers geographically nearby mirrors.

Managing repositories in DNF5

Under DNF5, repositories are managed directly through the built-in command branch. That replaces the historical Python plugins of older versions:

□□ Practical example:

List repositories, add new sources and enable them:

```
# Show all active repositories
dnf repo list

# Also show disabled sources
dnf repo list --all

# Add a new repository from a web source
sudo dnf config-manager addrepo --from-
repofile=https://download.docker.com/linux/fedora/docker-ce.repo

# Enable or disable a repository specifically
sudo dnf config-manager enable docker-ce-stable
sudo dnf config-manager disable docker-ce-test
```

Non-free software and multimedia: enabling RPM Fusion

The Fedora project is strictly committed to free software. Proprietary graphics drivers (such as Nvidia) or legally restricted video codecs (e.g. H.264/H.265 for video editing and media playback) are therefore not in the base system.

The Fedora community provides the external archive **RPM Fusion** for this, split into `free` (open-source software with patent restrictions) and `nonfree` (proprietary software such as graphics drivers):

□□ Practical example:

Enable RPM Fusion Free and Nonfree with one command:

```
# Enable RPM Fusion repositories for the current Fedora version
sudo dnf install \
  https://mirrors.rpmfusion.org/free/fedora/rpmfusion-free-release-$(rpm -E
```

```
%fedora).noarch.rpm \
  https://mirrors.rpmfusion.org/nonfree/fedora/rpmfusion-nonfree-release-$(rpm -E
%fedora).noarch.rpm
```

Community software with Fedora COPR

COPR (*Cool Other Package Repo*) is Fedora's counterpart to Ubuntu's PPAs or Arch's AUR. Developers and community members can build and publish their own packages there that have not yet been accepted into the official repositories:

□□ Practical example:

Search, enable and use a community repository from COPR:

```
# Enable a COPR repository for a desired tool
sudo dnf copr enable atim/lazygit

# Install the package as usual
sudo dnf install lazygit

# Disable a COPR repo that is no longer needed
sudo dnf copr disable atim/lazygit
```

> ⚠ **COPR security note:** COPR repositories are run by individuals and are not reviewed by the Fedora security team. Enable only sources from developers you trust.

Package management in daily work: search, install and remove

Daily console work is where DNF's commands are easiest to remember.

1. Searching and inspecting packages

Before you install software you usually want to know the exact package name and what it contains:

□□ Practical example:

Find packages in the software catalog and inspect details:

```
# Search names and descriptions for a term
dnf search webserver

# Show detailed metadata for a package (version, size, license)
dnf info nginx
```

2. The rescue command: dnf provides

A scenario every Linux admin knows: you follow a guide and the shell reports `semanage: command not found`. You do not know which RPM package contains that command.

This is where `dnf provides` (or `dnf whatprovides`) helps:

□□ Practical example:

Find which package provides a given file or command:

```
# Which package contains the htop command?
dnf provides /usr/bin/htop

# Which package provides the semanage tool?
dnf provides "*bin/semanage"
```

DNF searches the file lists of all repositories and reliably reports that in this case you need to install `policycoreutils-python-utils`.

3. Installing software

Installation uses the `install` subcommand. DNF computes all dependencies in advance and shows a detailed transaction summary:

□□ Practical example:

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

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

# Install several packages in a single transaction
sudo dnf install git tmux vim curl

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

Sample output of a DNF transaction summary:

```
Updating and loading repositories:
Repositories loaded.
Package            Arch     Version         Repository      Size
Installing:
 htop              x86_64   3.3.0-2.fc41    fedora         1.8 M
Installing dependencies:
 libnl3            x86_64   3.9.0-1.fc41    fedora       340.0 k

Transaction Summary:
```

```
 Installing:         2 packages
 Total size:         2.1 M
 Total download size: 1.2 M
Is this ok [y/N]:
```

> **Non-interactive mode for scripts:** When you write automation scripts or do not want to confirm the prompt manually with `y`, use the `-y` switch: `sudo dnf install -y htop`.

4. Uninstalling software and keeping the system clean

Packages are removed with `remove`:

Practical example:

Uninstall programs and find orphaned libraries:

```
# Remove a program
sudo dnf remove htop

# Automatically clean all unused, orphaned dependencies
sudo dnf autoremove
```

`dnf autoremove` is essential for system hygiene: it removes libraries that were once installed as a dependency for a program but are no longer needed by any installed software.

5. Managing package groups

If you want to set up a complete working environment (for example to compile software from source), you do not have to collect dozens of individual packages such as `gcc`, `make`, `autoconf` and `gdb` by hand:

Practical example:

Working with standardized Fedora package groups:

```
# List all available package groups
dnf group list

# Install the entire C/C++ development environment
sudo dnf group install "Development Tools"

# Remove a group again
sudo dnf group remove "Development Tools"
```

System updates and release upgrades

An up-to-date system is the most important foundation for IT security. DNF separates regular updates, isolated security updates and version upgrades.

Applying regular updates

□□ Practical example:

Check and install available system updates:

```
# Check which packages can be updated (without installing)
dnf check-update

# Bring all installed packages to the latest available state
sudo dnf upgrade
```

Applying targeted security updates

In production environments or on servers you often do not want to update the entire system, but only close known vulnerabilities (CVEs). In DNF5 this is controlled through the `advisory` subcommand (known as `updateinfo` in earlier DNF4 versions, which remains supported as a transparent compatibility alias):

□□ Practical example:

Check security advisories and apply them in a targeted way:

```
# Show a summary of all reported security issues (DNF5 default)
dnf advisory summary

# Show a detailed list with CVE numbers and severity
dnf advisory list --security

# Alternatively via the familiar compatibility alias from older DNF versions
dnf updateinfo summary

# Update only packages that close security issues
sudo dnf upgrade --security
```

Upgrading to a new Fedora major version

Fedora publishes a new major version every six months. The move (e.g. from Fedora 41 to Fedora 42) happens directly on the running system through the official `system-upgrade` procedure:

□□ Practical example:

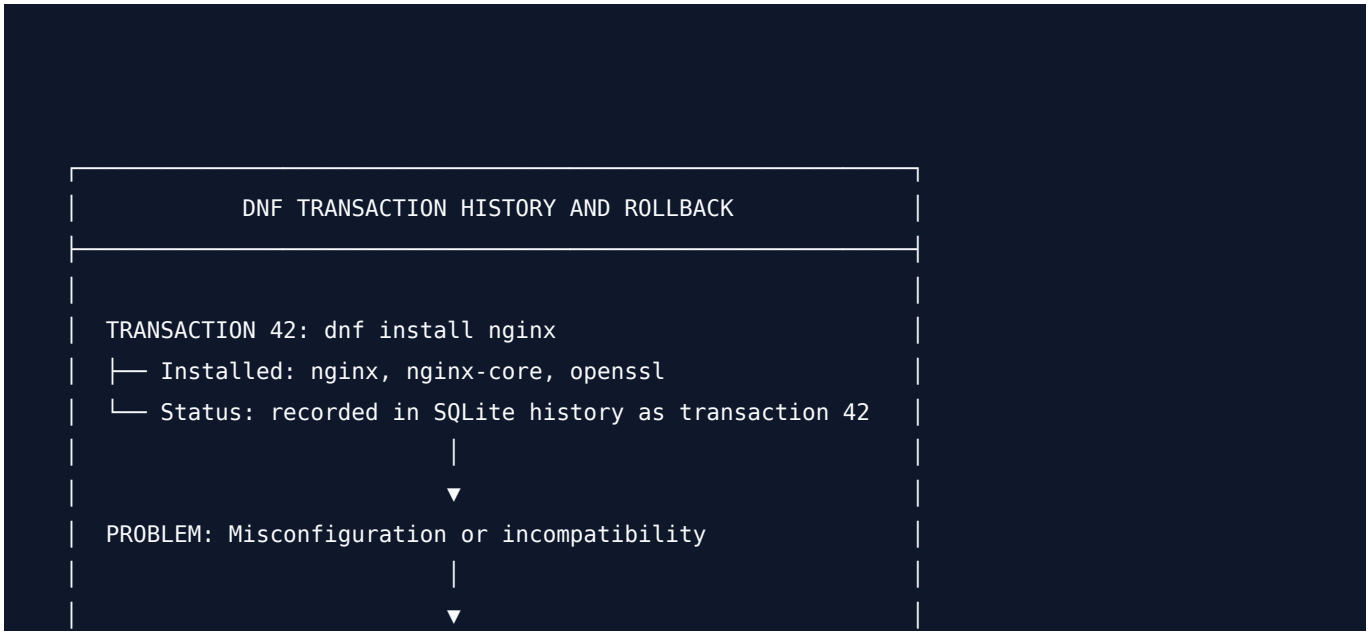
Full distribution upgrade in fail-safe offline mode:

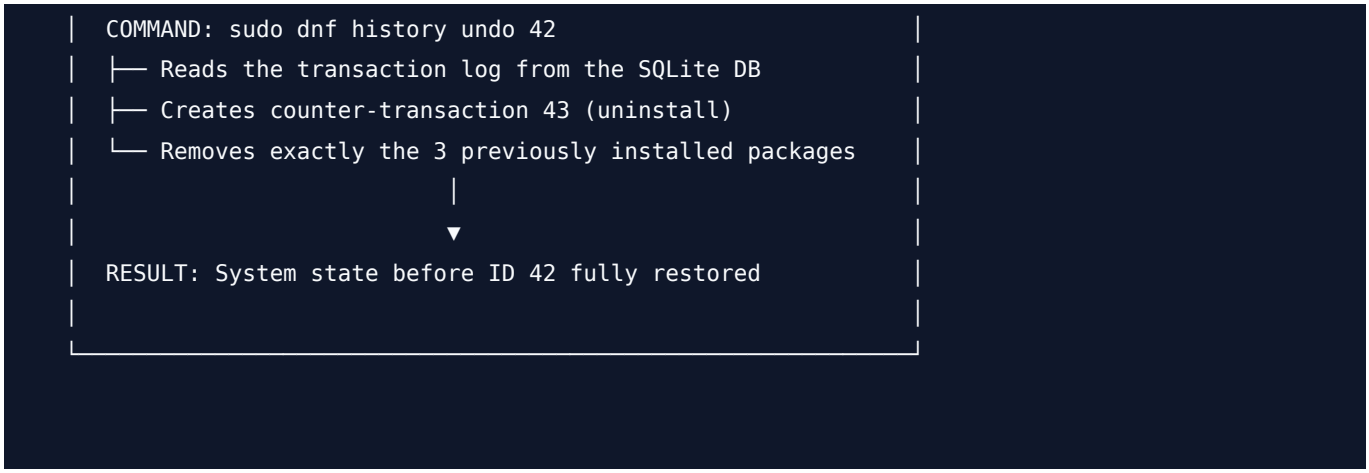
```
# 1. Fully update the existing system and refresh metadata
sudo dnf upgrade --refresh

# 2. Download all packages for the new target version in the background
sudo dnf system-upgrade download --releasever=42

# 3. Run the upgrade in an isolated maintenance reboot
sudo dnf system-upgrade reboot
```

The actual installation happens during the reboot in a minimal environment. That prevents running services or desktop sessions from crashing while central system libraries are swapped.

The time machine: transaction history and rollbacks

DNF's outstanding unique feature compared with tools such as APT is the complete **transaction history**. Every installation, every update and every uninstall is logged atomically in an SQLite database.





Analyzing transactions and undoing them

Imagine you installed a program that makes the system unstable or changed dependencies. Instead of reconstructing by hand which libraries were installed with it, you use the history:

□□ Practical example:

Inspect the transaction log, check details and undo transactions:

```
# List all past DNF actions
dnf history list

# Show in detail what happened in transaction 42
dnf history info 42

# Undo exactly this one transaction
sudo dnf history undo 42

# Reset the system to the state BEFORE a given transaction
sudo dnf history rollback 41
```

> ⚠ **Limits of rollback:** A `history rollback` can only succeed if all older RPM packages needed for the earlier state still exist in the repositories or in the local cache. On Fedora desktop systems, `undo` for individual transactions is the safer and more reliable choice.

Modern Fedora: when DNF, when Flatpak, when Toolbx?

On a modern Fedora Workstation, DNF does not exist in isolation. To keep the system stable you should know the division of labor:

Purpose	Recommended tool	Why?
System-near tools and CLI	**DNF / DNF5**	Shell tools (`tmux` , `git` , `htop`), server services and system libraries belong directly in the base system.
Graphical desktop apps	**Flatpak (Flathub)**	Browsers, messengers, image editors and office suites run isolated in a sandbox and do not endanger system libs.
Development environments	**Toolbx / Distrobox**	Keep compilers, Node.js versions or Python packages in OCI containers to keep the base system lean.

Troubleshooting and store hygiene

Even the best package manager occasionally needs administrative attention. Here are the four most important troubleshooting moves:

1. Cleaning a damaged metadata cache

When DNF reports that metadata is corrupt or checksums do not match:

□□ Practical example:

Empty the local cache and mirror metadata fresh from the servers:

```
# Empty the entire local package and metadata cache
sudo dnf clean all

# Rebuild the cache completely
sudo dnf check-update
```

2. Repairing GPG key validation

If DNF refuses to install a package because of an untrusted or expired GPG key, you can import the official keys by hand:

□□ Practical example:

Re-import official Fedora signing keys:

```
# Import official Fedora release keys
sudo rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$(rpm -E %fedora)-primary
```

3. Locking packages against updates (versionlock)

If you want to prevent a business-critical service or a specific driver from being updated during a general `dnf upgrade`, you can lock packages. In DNF5 the versionlock command is native in the `dnf5-plugins` package and stores exceptions in TOML under `/etc/dnf/versionlock.toml`:

□□ Practical example:

Pin packages with versionlock:

```
# Install the versionlock plugin (if not already present)
sudo dnf install dnf5-plugins

# Pin a package at the current version
sudo dnf versionlock add nginx

# List all locked packages (reads /etc/dnf/versionlock.toml)
dnf versionlock list

# Lift the lock again
sudo dnf versionlock delete nginx
```

4. Repairing RPM database inconsistencies

If the system crashed during a transaction and the package database is blocked:

□□ Practical example:

Check the package database for damage and repair it:

```
# Check for incomplete or duplicate packages
sudo dnf check

# Reindex the internal RPM database
sudo rpm --rebuilddb
```

Command Reference (Cheatsheet)

Category	Command	Function and purpose
Search	`dnf search <term>`	Searches package names and summaries
Info	`dnf info <package>`	Shows detailed metadata for version and license
File search	`dnf provides <path>`	Finds which package provides a given file
Install	`sudo dnf install <package>`	Installs packages including all dependencies
Uninstall	`sudo dnf remove <package>`	Uninstalls a package from the system
Cleanup	`sudo dnf autoremove`	Removes orphaned, unused dependencies
Update	`sudo dnf upgrade`	Updates all installed software packages
Advisories	`dnf advisory summary`	Lists security notices and CVEs (alias: `updateinfo`)
Security	`sudo dnf upgrade --security`	Applies security patches only
Versionlock	`sudo dnf versionlock add <pkg>`	Pins a package version in `/etc/dnf/versionlock.toml`
Groups	`sudo dnf group install <group>`	Installs preconfigured package bundles
History	`dnf history list`	Shows all past transactions with ID
Undo	`sudo dnf history undo <id>`	Undoes a transaction in a targeted way
Sources	`dnf repo list`	Lists all active repositories
Cache	`sudo dnf clean all`	Cleans local metadata and download caches

Further Resources

Resource	Description	Type

Official Fedora documentation	Full documentation for Fedora Workstation, Server and CoreOS	Official documentation
DNF5 project documentation	Official C++ architecture reference and manual of all DNF5 commands	Manual and reference
RPM Fusion portal	Official guides for multimedia codecs and Nvidia graphics drivers	Community portal
Fedora COPR build system	Central directory of community repositories and builds	Community platform
Fedora Packages Explorer	Interactive package search with changelogs, dependencies and versions	Official package search

Conclusion

With DNF and the modern DNF5 architecture, Fedora has one of the most advanced package-management systems in the Unix world. Where other distributions often need manual intervention on package conflicts, DNF stands out through the mathematical SAT solver (`libsolv`), extremely fast metadata handling in native C++ and a complete transaction history.

Anyone who has internalized the distinction between RPM files and repositories, tracks down missing commands with `dnf provides` , and treats the history with `undo` as a personal safety net, runs the system calmly and with control.

> **Practical tip for daily work:** Put `max_parallel_downloads=10` in `/etc/dnf/dnf.conf` . On larger system upgrades or when setting up a new machine, DNF downloads packages in parallel, which cuts waiting time drastically on fast internet links.