# Red Hat Enterprise Linux 9

## Managing software with the DNF tool

Managing content in the RPM repositories by using the DNF software management tool

Last Updated: 2025-05-20

# Red Hat Enterprise Linux 9 Managing software with the DNF tool

Managing content in the RPM repositories by using the DNF software management tool

## Legal Notice



## Abstract

Find, install, and utilize content distributed through the RPM repositories by using the DNF tool. Learn how to work with packages, modules, streams, and profiles.

# Table of Contents

# PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

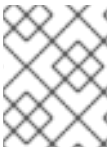
We appreciate your feedback on our documentation. Let us know how we can improve it.

**Submitting feedback through Jira (account required)**

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

# CHAPTER 1. SOFTWARE MANAGEMENT TOOLS IN RED HAT ENTERPRISE LINUX 9

In Red Hat Enterprise Linux (RHEL) 9, use the **DNF** utility to manage software. For compatibility reasons with previous major RHEL versions, you can still use the **yum** command. However, in RHEL 9, **yum** is an alias for **dnf** which provides a certain level of compatibility with **yum**.



NOTE

Although RHEL 8 and RHEL 9 are based on **DNF**, they are compatible with **YUM** used in RHEL 7.

# CHAPTER 2. DISTRIBUTION OF CONTENT IN RHEL 9

In the following sections, learn how the software is distributed in Red Hat Enterprise Linux 9.

## 2.1. REPOSITORIES

Red Hat Enterprise Linux (RHEL) distributes content through different repositories, for example:

**BaseOS**

Content in the BaseOS repository consists of the core set of the underlying operating system functionality that provides the foundation for all installations. This content is available in the RPM format and is subject to support terms similar to those in earlier releases of RHEL.

**AppStream**

Content in the AppStream repository includes additional user-space applications, runtime languages, and databases in support of the varied workloads and use cases.

**IMPORTANT**

Both the BaseOS and AppStream content sets are required by RHEL and are available in all RHEL subscriptions.

**CodeReady Linux Builder**

The CodeReady Linux Builder repository is available with all RHEL subscriptions. It provides additional packages for use by developers. Red Hat does not support packages included in the CodeReady Linux Builder repository.

**Additional resources**

- Package manifest

## 2.2. APPLICATION STREAMS

Red Hat provides multiple versions of user-space components as Application Streams, and they are updated more frequently than the core operating system packages. This provides more flexibility to customize Red Hat Enterprise Linux (RHEL) without impacting the underlying stability of the platform or specific deployments.

Application Streams are available in the following formats:

- RPM format
- Modules, which are an extension to the RPM format
- Software Collections

RHEL 9 improves Application Streams experience by providing initial Application Stream versions as RPMs, which you can install by using the **dnf install** command.

Starting with RHEL 9.1, Red Hat provides additional Application Stream versions as modules with a shorter life cycle.

**IMPORTANT**

Each Application Stream has its own life cycle, and it can be the same or shorter than the life cycle of RHEL 9. See Red Hat Enterprise Linux Application Streams Life Cycle .

Always determine which version of an Application Stream you want to install, and make sure to review the RHEL Application Stream life cycle first.

**Additional resources**

- Red Hat Enterprise Linux 9: Application Compatibility Guide
- Package manifest
- Red Hat Enterprise Linux Application Streams Life Cycle

## 2.3. MODULES

A module is a set of RPM packages that represent a component. A typical module contains the following package types:

- Packages with an application
- Packages with the application-specific dependency libraries
- Packages with documentation for the application
- Packages with helper utilities

## 2.4. MODULE STREAMS

Module streams are filters that can be imagined as virtual repositories in the AppStream physical repository. Module streams versions of the AppStream components. Each of the streams receives updates independently, and they can depend on other module streams.

Module streams can be active or inactive. Active streams give the system access to the RPM packages within the particular module stream, allowing the installation of the respective component version.

A stream is active in the following cases:

- If an administrator explicitly enables it.
- If the stream is a dependency of an enabled module.
- If the stream is the default stream. Each module can have a default stream but in Red Hat Enterprise Linux 9, no default streams are defined. If required, you can configure default streams as described in Defining custom default module streams and profiles .

Only one stream of a particular module can be active at a given point in time. Therefore, only packages from a particular stream are available.

Prior to selecting a particular stream for a runtime user application or a developer application, consider the following:

- Required functionality and which component versions support that functionality

- Compatibility with your application or use case
- The life cycle of the Application Stream and your update plan

For a list of all available modules and streams, see the Package manifest. For per-component changes, see the Release Notes.

**Additional resources**

- Modular dependencies and stream changes

## 2.5. MODULE PROFILES

A module profile is a list of recommended packages to be installed together for a particular use case such as for a server, client, development, minimal install, or other. These package lists can contain packages outside the module stream, usually from the BaseOS repository or the dependencies of the stream.

Installing packages by using a profile is a one-time action provided for the user's convenience. It is also possible to install packages by using multiple profiles of the same module stream without any further preparatory steps.

Each module stream can have any number of profiles, including none. For any given module stream, some of its profiles can be marked as *default* and are then used for profile installation actions if you did not explicitly specify a profile. However, the existence of a default profile for a module stream is not required.

**Example 2.1. nodejs module profiles**

The **nodejs** module, which provides the **Node.js** runtime environment, offers the following profiles for installation:

```
# dnf module list nodejs
Name     Stream Profiles                          Summary
nodejs   18     common [d], development, minimal, s2i   Javascript runtime
Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

In this example, the following profiles are available:

- **common**: The production-ready packages. This is the default profile ( **[d]**).
- **development**: The production-ready packages, including the **Node.js** development headers.
- **minimal**: The smallest set of packages that provides the **Node.js** runtime environment.
- **s2i**: Packages necessary for creating **Node.js** Source-to-Image (S2I) Linux containers.

# CHAPTER 3. CONFIGURING DNF

The configuration of **DNF** and related utilities is stored in the **[main]** section of the **/etc/dnf/dnf.conf** file.

## 3.1. VIEWING THE CURRENT DNF CONFIGURATIONS

The **[main]** section in the **/etc/dnf/dnf.conf** file contains only the settings that have been explicitly set. However, you can display all settings of the **[main]** section, including the ones that have not been set and which, therefore, use their default values.

### Procedure

- Display the global **DNF** configuration:

```
# dnf config-manager --dump
```

### Additional resources

- **dnf.conf(5)** man page on your system

## 3.2. SETTING DNF MAIN OPTIONS

The **/etc/dnf/dnf.conf** file contains one **[main]** section. The key-value pairs in this section affect how **DNF** operates and treats repositories.

### Procedure

1. Edit the **/etc/dnf/dnf.conf** file.
2. Update the **[main]** section according to your requirements.
3. Save the changes.

### Additional resources

- The **[main] OPTIONS** and **OPTIONS FOR BOTH [main] AND REPO** sections in the **dnf.conf(5)** man page on your system.

## 3.3. MANAGING DNF PLUG-INS

Every installed plug-in can have its own configuration file in the **/etc/dnf/plugins/** directory. Name plug-in configuration files in this directory ***<plug-in_name>*.conf**. By default, plug-ins are typically enabled. To disable a plug-in in one of these configuration files, add the following to the file:

```
[main]
enabled=False
```

## 3.4. ENABLING AND DISABLING DNF PLUG-INS

In the **DNF** tool, plug-ins are loaded by default. However, you can influence which plug-ins **DNF** loads.

**WARNING**

Disable all plug-ins only for diagnosing a potential problem. **DNF** requires certain plug-ins, such as **product-id** and **subscription-manager**, and disabling them causes Red Hat Enterprise Linux to not be able to install or update software from the Content Delivery Network (CDN).

**Procedure**

- Use one of the following methods to influence how **DNF** uses plug-ins:
  - To enable or disable loading of **DNF** plug-ins globally, add the **plugins** parameter to the **[main]** section of the **/etc/dnf/dnf.conf** file.
    - Set **plugins=1** (default) to enable loading of all **DNF** plug-ins.
    - Set **plugins=0** to disable loading of all **DNF** plug-ins.
  - To disable a particular plug-in, add **enabled=False** to the **[main]** section in the **/etc/dnf/plugins/*<plug-in_name>*.conf** file.
  - To disable all **DNF** plug-ins for a particular command, append the **--noplugins** option to the command. For example, to disable **DNF** plug-ins for a single update command, enter:

    ```
    # dnf --noplugins update
    ```

  - To disable certain **DNF** plug-ins for a single command, append the **--disableplugin=*plugin-name*** option to the command. For example, to disable a certain **DNF** plug-in for a single update command, enter:

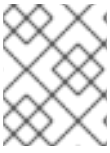
    ```
    # dnf update --disableplugin=<plugin_name>
    ```

  - To enable certain **DNF** plug-ins for a single command, append the **--enableplugin=*plugin-name*** option to the command. For example, to enable a certain **DNF** plug-in for a single update command, enter:

    ```
    # dnf update --enableplugin=<plugin_name>
    ```

## 3.5. EXCLUDING PACKAGES FROM DNF OPERATIONS

You can configure **DNF** to exclude packages from any **DNF** operation by using the **excludepkgs** option. You can define **excludepkgs** in the **[main]** or the repository section of the **/etc/dnf/dnf.conf** file.



**NOTE**

You can temporarily disable excluding the configured packages from an operation by using the **--disableexcludes** option.

**Procedure**

- Exclude packages from the **DNF** operation by adding the following line to the **/etc/dnf/dnf.conf** file:

```
excludepkgs=<package_name_1>,<package_name_2> ...
```

  Alternatively, use global expressions instead of package names to define packages you want to exclude. For more information, see Specifying global expressions in DNF input .

**Additional resources**

- **dnf.conf(5)** man page on your system
- Specifying global expressions in DNF input

# CHAPTER 4. SEARCHING FOR RHEL 9 CONTENT

In the following sections, learn how to locate and examine content in the AppStream and BaseOS repositories in Red Hat Enterprise Linux 9 by using **DNF**.

## 4.1. SEARCHING FOR SOFTWARE PACKAGES

To identify which package provides the software you require, you can use **DNF** to search the repositories.

**Procedure**

- Depending on your scenario, use one of the following options to search the repository:
  - To search for a term in the name or summary of packages, enter:

    ```
    $ dnf search <term>
    ```

  - To search for a term in the name, summary, or description of packages, enter:

    ```
    $ dnf search --all <term>
    ```

    Note that searching additionally in the description by using the **--all** option is slower than a normal search operation.

  - To search for a package name and list the package name and its version in the output, enter:

    ```
    $ dnf repoquery <package_name>
    ```

  - To search for which package provides a file, specify the file name or the path to the file:

    ```
    $ dnf provides <file_name>
    ```

## 4.2. LISTING SOFTWARE PACKAGES

You can use **DNF** to display a list of packages and their versions that are available in the repositories. If required, you can filter this list and, for example, only list packages for which updates are available.

**Procedure**

- List the latest versions of all available packages, including architectures, version numbers, and the repository they where installed from:

  ```
  $ dnf list --all
  ...
  zlib.x86_64          1.2.11-39.el9   @rhel-9-for-x86_64-baseos-rpms
  zlib.i686            1.2.11-39.el9   rhel-9-for-x86_64-baseos-rpms
  zlib-devel.i686      2.11-39.el9     rhel-9-for-x86_64-appstream-rpms
  zlib-devel.x86_64    1.2.11-39.el9   rhel-9-for-x86_64-appstream-rpms
  ...
  ```

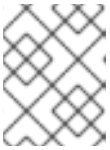
  The @ sign in front of a repository indicates that the package in this line is currently installed.

Alternatively, to display all available packages, including version numbers and architectures, enter:

```
$ dnf repoquery
...
zlib-0:1.2.11-35.el9_1.i686
zlib-0:1.2.11-35.el9_1.x86_64
zlib-0:1.2.11-39.el9.i686
zlib-0:1.2.11-39.el9.x86_64
zlib-devel-0:1.2.11-39.el9.i686
zlib-devel-0:1.2.11-39.el9.x86_64
...
```

Optionally, you can filter the output by using other options instead of **--all**, for example:

- Use **--installed** to list only installed packages.
- Use **--available** to list all available packages.
- Use **--upgrades** to list packages for which newer versions are available.



NOTE

You can filter the results by appending global expressions as arguments. For more details, see Specifying global expressions in DNF input .

## 4.3. LISTING REPOSITORIES

To get an overview of repositories that are enabled and disabled on your system, you can list them.

**Procedure**

1. List all enabled repositories on your system:

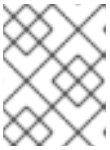
```
$ dnf repolist
```

To display only certain repositories, append one of the following options to the command:

- Append **--disabled** to list only disabled repositories.
- Append **--all** to list both enabled and disabled repositories.

2. Optional: List additional information about the repositories:

```
$ dnf repoinfo <repository_name>
```



NOTE

You can filter the results by using global expressions. For details, see Specifying global expressions in DNF input.

## 4.4. DISPLAYING PACKAGE INFORMATION

You can query **DNF** repositories to display further details about a package, such as the following:

- Version
- Release
- Architecture
- Package size
- Description

**Procedure**

- Display information about one or more available packages:

```
$ dnf info <package_name>
```

This command displays the information for the currently installed package and, if available, its newer versions that are in the repository. Alternatively, use the following command to display the information for all packages with the specified name in the repository:

```
$ dnf repoquery --info <package_name>
```

NOTE

You can filter the results by appending global expressions as arguments. For details, see Specifying global expressions in DNF input .

## 4.5. LISTING PACKAGE GROUPS AND PACKAGES THEY PROVIDE

Package groups bundle multiple packages, and you can use package groups to install all packages assigned to a group in a single step. However, before the installation, you must identify the name of the required package group.

**Procedure**

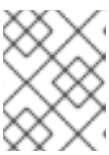
1. List both installed and available groups:

```
$ dnf group list
```

Note that you can filter the results by appending the **--installed** and **--available** option to the **dnf group list** command. By using the **--hidden** option, you can display hidden groups in the output.

2. List mandatory, optional, and default packages contained in a particular group:

```
$ dnf group info "<group_name>"
```



NOTE

You can filter the results by appending global expressions as arguments. For more details, see Specifying global expressions in DNF input .

3. Optional: View the number of installed and available groups:

```
$ dnf group summary
```

## 4.6. LISTING AVAILABLE MODULES AND THEIR CONTENTS

By searching for modules and displaying information about them with **DNF**, you can identify which modules are available in the repositories and select the appropriate stream before you install a module.

**Procedure**

1. List the module information in one of the following ways:

   - List all available modules:

     ```
     $ dnf module list
     Name        Stream   Profiles                              Summary
     ...
     nodejs      18       common [d], development, minimal, s2i  Javascript runtime
     postgresql  15       client, server                         PostgreSQL server and client module
     ...
     Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
     ```

     Use the **dnf module list *<module_name>*** command to list the same information but only for a specific module.

   - Search for which module provides a certain package:

     ```
     $ dnf module provides <package_name>
     ```

     For example, to display which module and profiles provide the **npm** package, enter:

     ```
     # dnf module provides npm
     npm-1:8.19.2-1.18.10.0.3.module+el9.1.0+16866+0fab0697.x86_64
     Module   : nodejs:18:9010020221009220316:rhel9:x86_64
     Profiles : common development s2i
     Repo     : rhel-9-for-x86_64-appstream-rpms
     Summary  : Javascript runtime
     ...
     ```

2. Use one of these methods to list module details:

   - List all details about a module, including a description, list of all profiles, and a list of all packages the module provides:

     ```
     $ dnf module info <module_name>
     ```

     For example, to display details about the **nodejs** package, enter:

     ```
     $ dnf module info nodejs
     Name           : nodejs
     Stream         : 18
     Version        : 9010020221009220316
     Context        : rhel9
     ```

```
Architecture     : x86_64
Profiles         : common [d], development, minimal, s2i
Default profiles : common
Repo             : rhel-9-for-x86_64-appstream-rpms
Summary          : Javascript runtime
Description      : Node.js is a platform built on Chrome's JavaScript runtime...
Requires         : platform:[el9]
Artifacts        : nodejs-1:18.10.0-3.module+el9.1.0+16866+0fab0697.src
                 : nodejs-1:18.10.0-3.module+el9.1.0+16866+0fab0697.x86_64
                 : npm-1:8.19.2-1.18.10.0.3.module+el9.1.0+16866+0fab0697.x86_64
...
```

- List which packages each module profile installs:

```
$ dnf module info --profile <module_name>
```

For example, to display this information for the **nodejs** module, enter:

```
$ dnf module info --profile nodejs
Name        : nodejs:18:9010020221009220316:rhel9:x86_64
common      : nodejs
            : npm
development : nodejs
            : nodejs-devel
            : npm
minimal     : nodejs
s2i         : nodejs
            : nodejs-nodemon
            : npm
...
```

**Additional resources**

- Modules
- Module streams
- Module profiles

## 4.7. SPECIFYING GLOBAL EXPRESSIONS IN DNF INPUT

You can filter the results of **dnf** commands by appending one or more global expressions as arguments.

**Procedure**

- Use one of the following methods if you use global expressions in **dnf** commands:
  - Enclose the entire global expression in single or double quotation marks:

```
# dnf provides "*/<file_name>"
```

Note that you must precede ***<file_name>*** either by / for an absolute path or */ to use a wildcard if the full path is unknown.

- Escape the wildcard characters by preceding them with a backslash (\) character:

```
# dnf provides \*/<file_name>
```

## 4.8. ADDITIONAL RESOURCES

- Commands for listing content in RHEL 9

# CHAPTER 5. INSTALLING RHEL 9 CONTENT

In the following sections, learn how to install content in Red Hat Enterprise Linux 9 by using **DNF**.

## 5.1. INSTALLING PACKAGES

If a software is not part of the default installation, you can manually install it. **DNF** automatically resolves and installs dependencies.

### Prerequisites

- Optional: You know the name of the package you want to install .
- If the package you want to install is provided by a module stream, the respective module stream is enabled.

### Procedure

- Use one of the following methods to install packages:
  - To install packages from the repositories, enter:

    ```
    # dnf install <package_name_1> <package_name_2> ...
    ```

    If you install packages on a system that supports multiple architectures, such as **i686** and **x86_64**, you can specify the architecture of the package by appending it to the package name:

    ```
    # dnf install <package_name>.<architecture>
    ```

  - To install a package if you only know the path to the file the package provides but not the package name, you can use this path to install the corresponding package:

    ```
    # dnf install <path_to_file>
    ```

  - To install a local RPM file, enter:

    ```
    # dnf install <path_to_RPM_file>
    ```

    If the package has dependencies, specify the paths to these RPM files as well. Otherwise, **DNF** downloads the dependencies from the repositories or fails if they are not available in the repositories.

### Additional resources

- Installing modular content

## 5.2. INSTALLING PACKAGE GROUPS

Package groups bundle multiple packages, and you can use package groups to install all packages assigned to a group in a single step.

**Prerequisites**

- You know the name or ID of the group you want to install .

**Procedure**

- Install a package group:

```
# dnf group install <group_name_or_ID>
```

## 5.3. INSTALLING MODULAR CONTENT

For certain software, Red Hat provides modules. You can use modules to install a specific version (stream) and set of packages (profiles).

**Procedure**

1. List modules that provide the package you want to install:

```
# dnf module list <module_name>
```

For example, to list the details about the **nodejs** module, enter:

```
# dnf module list nodejs
Name     Stream   Profiles                                Summary
nodejs   18       common [d], development, minimal, s2i   Javascript runtime
nodejs   ...      common [d], development, minimal, s2i   Javascript runtime

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

2. Install a module:

```
# dnf module install <module_name>:<stream>/<profile>
```

If a default profile for a stream is defined, you can omit **/<profile>** in the command to install this default profile of the stream.

NOTE

In Red Hat Enterprise Linux 9, no default module streams are predefined. However, if you specify the stream during the module installation as shown, you do not have to manually enable the stream in advance.

For example, to install the default profile (**common**) from stream **18** of the **nodejs** module, enter:

```
# dnf module install nodejs:18
================================================================================
 Package                    Architecture   Version   Repository                Size
================================================================================
Installing group/module packages:
```

```
 nodejs                  x86_64        ...      rhel-9-for-x86_64-appstream-rpms    12 M
 npm                     x86_64        ...      rhel-9-for-x86_64-appstream-rpms   2.5 M
Installing weak dependencies:
 nodejs-docs             noarch        ..       rhel-9-for-x86_64-appstream-rpms   7.6 M
 nodejs-full-i18n        x86_64        ..       rhel-9-for-x86_64-appstream-rpms   8.4 M
Installing module profiles:
 nodejs/common
Enabling module streams:
 nodejs                                18
```

### Verification

- Verify that the correct module stream is enabled (**[e]**) and the required profile was installed ( **[i]**):

```
# dnf module list nodejs
Updating Subscription Management repositories.
Last metadata expiration check: 0:33:24 ago on Mon 24 Jul 2023 04:59:01 PM CEST.
Red Hat Enterprise Linux 9 for x86_64 - AppStream (RPMs)
Name     Stream   Profiles                                Summary
nodejs   18 [e]   common [d] [i], development, minimal, s2i   Javascript runtime
...

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

### Additional resources

- Modules
- Module streams
- Module profiles

## 5.4. DEFINING CUSTOM DEFAULT MODULE STREAMS AND PROFILES

Red Hat Enterprise Linux 9 does not define default streams in the AppStream repository. However, you can configure a default module stream and default module profile. In this case, you can omit this information when you install the default stream and profile of a module.

### Procedure

1. Use the **dnf module list *<module_name>*** command to display the available streams and their profiles, for example:

```
# dnf module list nodejs
Name     Stream   Profiles                                Summary
nodejs   18       common [d], development, minimal, s2i   Javascript runtime
```

   In this example, **nodejs:18** is not set as the default stream, and the default profile in this stream is **common**.

2. Create a YAML file in the **/etc/dnf/modules.defaults.d/** directory to define the default stream and profile for a module.

For example, create the **/etc/dnf/modules.defaults.d/nodejs.yaml** file with the following content to define **18** as the default stream and **minimal** as the default profile for the **nodejs** module:

```
document: modulemd-defaults
version: 1
data:
  module: nodejs
  stream: "18"
  profiles:
    '18': [minimal]
```

**Verification**

- Use the **dnf module list *<module_name>*** command to verify the new default stream and profile settings, for example:

```
# dnf module list nodejs
Name     Stream   Profiles                                Summary
nodejs   18 [d]   common, development, minimal [d], s2i   Javascript runtime
```

**Additional resources**

- Modules
- Module streams
- Module profiles

## 5.5. ADDITIONAL RESOURCES

- Commands for installing content in RHEL 9

# CHAPTER 6. UPDATING RHEL 9 CONTENT

With **DNF**, you can check if your system has any pending updates. You can list packages that need updating and choose to update a single package, multiple packages, or all packages at once. If any of the packages you choose to update have dependencies, these dependencies are updated as well.

## 6.1. CHECKING FOR UPDATES

To identify which packages installed on your system have available updates, you can list them.

**Procedure**

- Check the available updates for installed packages:

```
# dnf check-update
```

  The output returns the list of packages and their dependencies that have an update available.

## 6.2. UPDATING PACKAGES

You can use **DNF** to update a single package, a package group, or all packages and their dependencies at once.

IMPORTANT

When applying updates to the kernel, **dnf** always installs a new kernel regardless of whether you are using the **dnf upgrade** or **dnf install** command. Note that this only applies to packages identified by using the **installonlypkgs** DNF configuration option. Such packages include, for example, the **kernel**, **kernel-core**, and **kernel-modules** packages.

**Procedure**

- Depending on your scenario, use one of the following options to apply updates:
  - To update all packages and their dependencies, enter:

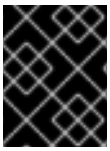
```
# dnf upgrade
```

  - To update a single package, enter:

```
# dnf upgrade <package_name>
```

  - To update packages only from a specific package group, enter:

```
# dnf group upgrade <group_name>
```



IMPORTANT

If you upgraded the GRUB boot loader packages on a BIOS or IBM Power system, reinstall GRUB. See Reinstalling GRUB.

## 6.3. UPDATING SECURITY-RELATED PACKAGES

You can use **DNF** to update security-related packages.

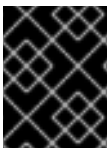
**Procedure**

- Depending on your scenario, use one of the following options to apply updates:
  - To upgrade to the latest available packages that have security errata, enter:

```
# dnf upgrade --security
```

  - To upgrade to the last security errata packages, enter:

```
# dnf upgrade-minimal --security
```



**IMPORTANT**

If you upgraded the GRUB boot loader packages on a BIOS or IBM Power system, reinstall GRUB. See Reinstalling GRUB.

**Additional resources**

- Managing and monitoring security updates

# CHAPTER 7. AUTOMATING SOFTWARE UPDATES IN RHEL 9

**DNF Automatic** is an alternative command-line interface to **DNF** that is suited for automatic and regular execution by using systemd timers, cron jobs, and other such tools.

**DNF Automatic** synchronizes package metadata as needed, checks for updates available, and then performs one of the following actions depending on how you configure the tool:

- Exit
- Download updated packages
- Download and apply the updates

The outcome of the operation is then reported by a selected mechanism, such as the standard output or email.

## 7.1. INSTALLING DNF AUTOMATIC

To check and download package updates automatically and regularly, you can use the **DNF Automatic** tool that is provided by the **dnf-automatic** package.

**Procedure**

- Install the **dnf-automatic** package:

```
# dnf install dnf-automatic
```

**Verification**

- Verify the successful installation by confirming the presence of the **dnf-automatic** package:

```
# rpm -qi dnf-automatic
```

## 7.2. DNF AUTOMATIC CONFIGURATION FILE

By default, **DNF Automatic** uses **/etc/dnf/automatic.conf** as its configuration file to define its behavior.

The configuration file is separated into the following topical sections:

- **[commands]**
  Sets the mode of operation of **DNF Automatic**.

**WARNING**

Settings of the operation mode from the **[commands]** section are overridden by settings used by a systemd timer unit for all timer units except **dnf-automatic.timer**.

- **[emitters]**
  Defines how the results of **DNF Automatic** are reported.
- **[command]**
  Defines the command emitter configuration.
- **[command_email]**
  Provides the email emitter configuration for an external command used to send email.
- **[email]**
  Provides the email emitter configuration.
- **[base]**
  Overrides settings from the main configuration file of **DNF**.

With the default settings of the **/etc/dnf/automatic.conf** file, **DNF Automatic** checks for available updates, downloads them, and reports the results to standard output.

### Additional resources

- **dnf-automatic(8)** man page on your system
- Overview of the systemd timer units included in the dnf-automatic package

## 7.3. ENABLING DNF AUTOMATIC

To run **DNF Automatic** once, you must start a systemd timer unit. However, if you want to run **DNF Automatic** periodically, you must enable the timer unit. You can use one of the timer units provided in the **dnf-automatic** package, or you can create a drop-in file for the timer unit to adjust the execution time.

### Prerequisites

- You specified the behavior of **DNF Automatic** by modifying the **/etc/dnf/automatic.conf** configuration file.

### Procedure

- Enable and execute a systemd timer unit immediately:

  ```
  # systemctl enable --now <timer_name>
  ```

  If you want to only enable the timer without executing it immediately, omit the **--now** option.

  You can use the following timers:

  - **dnf-automatic-download.timer**: Downloads available updates.
  - **dnf-automatic-install.timer**: Downloads and installs available updates.
  - **dnf-automatic-notifyonly.timer**: Reports available updates.
  - **dnf-automatic.timer**: Downloads, downloads and installs, or reports available updates.

### Verification

- Verify that the timer is enabled:

```
# systemctl status <timer_name>
```

- Optional: Check when each of the timers on your system ran the last time:

```
# systemctl list-timers --all
```

**Additional resources**

- **dnf-automatic(8)** man page on your system
- Overview of the systemd timer units included in the dnf-automatic package

## 7.4. OVERVIEW OF THE SYSTEMD TIMER UNITS INCLUDED IN THE DNF-AUTOMATIC PACKAGE

The systemd timer units take precedence and override the settings in the **/etc/dnf/automatic.conf** configuration file when downloading and applying updates.

For example, if you set **download_updates = yes** in the **/etc/dnf/automatic.conf** configuration file, but you have activated the **dnf-automatic-notifyonly.timer unit**, the packages will not be downloaded.

**Table 7.1. systemd timers included in the dnf-automatic package**

| Timer unit | Function | Overrides the **apply_updates** and **download_updates** settings in the **[commands]** section of the **/etc/dnf/automatic.conf** file? |
|---|---|---|
| **dnf-automatic-download.timer** | Downloads packages to cache and makes them available for updating.<br><br>This timer unit does not install the updated packages. To perform the installation, you must run the **dnf update** command. | Yes |
| **dnf-automatic-install.timer** | Downloads and installs updated packages. | Yes |
| **dnf-automatic-notifyonly.timer** | Downloads only repository data to keep the repository cache up-to-date and notifies you about available updates.<br><br>This timer unit does not download or install the updated packages. | Yes |

| Timer unit | Function | Overrides the **apply_updates** and **download_updates** settings in the **[commands]** section of the **/etc/dnf/automatic.conf** file? |
|---|---|---|
| **dnf-automatic.timer** | The behavior of this timer when downloading and applying updates is specified by the settings in the **/etc/dnf/automatic.conf** configuration file.<br><br>This timer downloads packages, but does not install them. | No |

# CHAPTER 8. REMOVING RHEL 9 CONTENT

In the following sections, learn how to remove content in Red Hat Enterprise Linux 9 by using **DNF**.

## 8.1. REMOVING INSTALLED PACKAGES

You can use **DNF** to remove a single package or multiple packages installed on your system. If any of the packages you choose to remove have unused dependencies, **DNF** uninstalls these dependencies as well.

**Procedure**

- Remove particular packages:

```
# dnf remove <package_name_1> <package_name_2> ...
```

## 8.2. REMOVING PACKAGE GROUPS

Package groups bundle multiple packages. You can use package groups to remove all packages assigned to a group in a single step.

**Procedure**

- Remove package groups by the group name or group ID:

```
# dnf group remove <group_name> <group_id>
```

## 8.3. REMOVING INSTALLED MODULAR CONTENT

When removing installed modular content, you can remove packages from either a selected profile or the whole stream.

IMPORTANT

**DNF** tries to remove all packages with a name corresponding to the packages installed with a profile or a stream, including their dependent packages. Always check the list of packages to be removed before you proceed, especially if you have enabled custom repositories on your system.

### 8.3.1. Removing packages from an installed profile

When you remove packages installed with a profile, all packages with a name corresponding to the packages installed by the profile are removed. This includes their dependencies, with the exception of packages required by a different profile.

To remove all packages from a selected stream, complete the steps in Removing all packages from a module stream.

**Prerequisites**

- The selected profile is installed by using the **dnf module install *<module-name:stream/profile>*** command or as a default profile by using the **dnf install *<module-name:stream>*** command.

**Procedure**

- Uninstall packages that belong to the selected profile:

```
# dnf module remove <module-name:stream/profile>
```

For example, to remove packages and their dependencies from the **development** profile of the **nodejs:18** module stream, enter:

```
# dnf module remove nodejs:18/development
(...)
Dependencies resolved.
================================================================================

Package          Architecture   Version
  Repository       Size
================================================================================

Removing:
nodejs-devel     x86_64         1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
  @rhel-AppStream  950 k
Removing unused dependencies:
brotli           x86_64         1.0.9-6.el9
  @rhel-AppStream  754 k
brotli-devel     x86_64         1.0.9-6.el9
  @rhel-AppStream  55 k
...
Disabling module profiles:
nodejs/development

Transaction Summary
================================================================================

Remove  26 Packages

Freed space: 8.3 M
Is this ok [y/N]: y
```

**WARNING**

Check the list of packages under **Removing:** and **Removing unused dependencies:** before you proceed with the removal transaction. This transaction removes requested packages, unused dependencies, and dependent packages, which might result in the system failure.

Alternatively, uninstall packages from all installed profiles within a stream:

```
# dnf module remove module-name:stream
```

NOTE

These operations will not remove packages from the stream that do not belong to any of the profiles.

**Verification**

- Verify that the correct profile was removed:

```
$ dnf module info nodejs
...
Name            : nodejs
Stream          : 18 [e] [a]
Version         : 9010020221009220316
Context         : rhel9
Architecture    : x86_64
Profiles        : common [d] [i], development, minimal [i], s2i [i]
Default profiles : common
Repo            : rhel-AppStream
Summary         : Javascript runtime
...
Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled, [a]ctive
```

All profiles except **development** are currently installed ( **[i]**).

**Additional resources**

- Modular dependencies and stream changes

## 8.3.2. Removing all packages from a module stream

When you remove packages installed with a module stream, all packages with a name corresponding to the packages installed by the stream are removed. This includes their dependencies, with the exception of packages required by other modules.

To remove only packages from a selected profile, complete the steps in Removing packages from an installed profile.

**Prerequisites**

- The module stream is enabled and at least some packages from the stream are installed.

**Procedure**

1. Remove all packages from a selected stream:

```
# dnf module remove --all <module_name:stream>
```

For example, to remove all packages from the **nodejs:18** module stream, enter:

```
# dnf module remove --all nodejs:18
(...)
Dependencies resolved.
================================================================================
```

```
========
Package         Architecture   Version
 Repository      Size
================================================================================
========
Removing:
nodejs          x86_64         1:18.10.0-3.module+el9.1.0+16866+0fab0697
 @rhel-AppStream  43 M
nodejs-devel    x86_64         1:18.10.0-3.module+el9.1.0+16866+0fab0697
 @rhel-AppStream  953 k
nodejs-docs     noarch         1:18.10.0-3.module+el9.1.0+16866+0fab0697
 @rhel-AppStream  78 M
nodejs-full-i18n x86_64        1:18.10.0-3.module+el9.1.0+16866+0fab0697
 @rhel-AppStream  29 M
nodejs-nodemon  noarch         2.0.15-1.module+el9.1.0+15718+e52ec601
 @rhel-AppStream  2.0 M
nodejs-packaging noarch        2021.06-4.module+el9.1.0+15718+e52ec601
 @rhel-AppStream  41 k
npm             x86_64         1:8.19.2-1.18.10.0.3.module+el9.1.0+16866+0fab0697
 @rhel-AppStream  6.9 M
Removing unused dependencies:
brotli          x86_64         1.0.9-6.el9
 @rhel-AppStream  754 k
brotli-devel    x86_64         1.0.9-6.el9
 @rhel-AppStream  55 k
...
Disabling module profiles:
nodejs/common
nodejs/development
nodejs/minimal
nodejs/s2i


Transaction Summary
================================================================================
========
Remove  31 Packages

Freed space: 167 M
Is this ok [y/N]: y
```

**WARNING**

Check the list of packages under **Removing:** and **Removing unused dependencies:** before you proceed with the removal transaction. This transaction removes requested packages, unused dependencies, and dependent packages, which might result in the system failure.

2. Optional: Reset or disable the stream by entering one of the following commands:

```
# dnf module reset <module_name>
# dnf module disable <module_name>
```

**Verification**

- Verify that all packages from the selected module stream were removed:

```
$ dnf module info nodejs
...
Name             : nodejs
Stream           : 18 [e] [a]
Version          : 9010020221009220316
Context          : rhel9
Architecture     : x86_64
Profiles         : common [d], development, minimal, s2i
Default profiles : common
...
Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled, [a]ctive
```

**Additional resources**

- Modular dependencies and stream changes
- Resetting module streams
- Disabling all streams of a module

## 8.4. ADDITIONAL RESOURCES

- Commands for removing content in RHEL 9

# CHAPTER 9. HANDLING PACKAGE MANAGEMENT HISTORY

With the **dnf history** command, you can review the following information:

- Timeline of **DNF** transactions.
- Dates and times the transactions occurred.
- Number of packages affected by the transactions.
- Whether the transactions succeeded or were aborted.
- If the RPM database was changed between the transactions.

You can also use the **dnf history** command to undo the transactions.

## 9.1. LISTING TRANSACTIONS

You can use **DNF** to perform the following tasks:

- List the latest transactions.
- List the latest operations for a selected package.
- Display details of a particular transaction.

**Procedure**

- Depending on your scenario, use one of the following options to display transaction information:
  - To display a list of all the latest **DNF** transactions, enter:

    ```
    # dnf history
    ```

    The output contains the following information:

    - The **Action(s)** column displays which type of action was performed during a transaction, for example, Install (**I**), Upgrade (**U**), Remove (**E**), and other actions.
    - The **Altered** column displays the number of actions performed during the transaction. The number of actions can also be followed by the result of the transaction. For more information about the values of the **Action(s)** and **Altered** columns, see the **dnf(8)** man page on your system.
  - To display a list of all the latest operations for a selected package, enter:

    ```
    # dnf history list <package_name>
    ```

  - To display details of a particular transaction, enter:

    ```
    # dnf history info <transaction_id>
    ```

NOTE

You can filter the results by appending global expressions as arguments. For more details, see Specifying global expressions in dnf input .

**Additional resources**

- **dnf(8)** man page on your system

## 9.2. REVERTING DNF TRANSACTIONS

Reverting a **DNF** transaction can be useful if you want to undo operations performed during the transaction. For example, if you installed several packages by using the **dnf install** command, you can uninstall these packages at once by reverting an installation transaction.

You can revert **DNF** transactions the following ways:

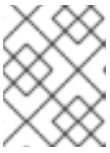
- Revert a single **DNF** transaction by using the **dnf history undo** command.
- Revert all **DNF** transactions performed between the specified transaction and the last transaction by using the **dnf history rollback** command.

Downgrading RHEL system packages to an older version by using the **dnf history undo** and **dnf history rollback** command is not supported. This concerns especially the **selinux**, **selinux-policy-***, **kernel**, and **glibc** packages, and dependencies of **glibc** such as **gcc**. Therefore, downgrading a system to a minor version (for example, from RHEL 9.1 to RHEL 9.0) is not recommended because it might leave the system in an incorrect state.

### 9.2.1. Reverting a single DNF transaction

You can revert steps performed within a single transaction by using the **dnf history undo** command:

- If the transaction installed a new package, **dnf history undo** uninstalls the package.
- If the transaction uninstalled a package, **dnf history undo** reinstalls the package.
- The **dnf history undo** command also attempts to downgrade all updated packages to their previous versions if the older packages are still available.



NOTE

If an older package version is not available, the downgrade by using the **dnf history undo** command fails.

**Procedure**

1. Identify the ID of a transaction you want to revert:

```
# dnf history
ID | Command line     | Date and time     | Action(s)      | Altered
-------------------------------------------------------------------------
13 | install zip      | 2022-11-03 10:49  | Install        |    1
12 | install unzip    | 2022-11-03 10:49  | Install        |    1
```

2. Optional: Verify that this is the transaction you want to revert by displaying its details:

```
# dnf history info <transaction_id>
```

3. Revert the transaction:

```
# dnf history undo <transaction_id>
```

For example, if you want to uninstall the previously installed **unzip** package, enter:

```
# dnf history undo 12
```

### 9.2.2. Reverting multiple DNF transactions

You can revert all **DNF** transactions performed between a specified transaction and the last transaction by using the **dnf history rollback** command. Note that the transaction specified by the transaction ID remains unchanged.

**Procedure**

1. Identify the transaction ID of the state you want to revert to:

```
# dnf history
ID | Command line     | Date and time     | Action(s)   | Altered
-------------------------------------------------------------------
14 | install wget     | 2022-11-03 10:49  | Install     |    1
13 | install unzip    | 2022-11-03 10:49  | Install     |    1
12 | install vim-X11  | 2022-11-03 10:20  | Install     |  171 EE
```

2. Revert specified transactions:

```
# dnf history rollback <transaction_id>
```

For example, to revert to the state before the **wget** and **unzip** packages were installed, enter:

```
# dnf history rollback 12
```

Alternatively, to revert all transactions in the transaction history, use the transaction ID **1**:

```
# dnf history rollback 1
```

# CHAPTER 10. MANAGING CUSTOM SOFTWARE REPOSITORIES

You can configure a repository in the **/etc/dnf/dnf.conf** file or in a **.repo** file in the **/etc/yum.repos.d/** directory.

IMPORTANT

Define your repositories in the **.repo** file instead of **/etc/dnf/dnf.conf**.

The **/etc/dnf/dnf.conf** file contains the **[main]** section and can contain one or more repository sections (**[*<repository-ID>*]**) that you can use to set repository-specific options. The values you define in individual repository sections of the **/etc/dnf/dnf.conf** file override values set in the **[main]** section.

## 10.1. DNF REPOSITORY OPTIONS

The **/etc/dnf/dnf.conf** configuration file contains repository sections with a unique repository ID in brackets (**[]**). You can use such sections to define individual **DNF** repositories.

IMPORTANT

Repository IDs in **[]** must be unique.

For a complete list of available repository ID options, see the **[*<repository-ID>*] OPTIONS** section of the **dnf.conf(5)** man page on your system.

## 10.2. ADDING A DNF REPOSITORY

You can add a **DNF** repository to your system by using the **dnf config-manager --add-repo** command.

**Procedure**

1. Add a repository to your system:

```
# dnf config-manager --add-repo <repository_URL>
```

   Note that repositories added by this command are enabled by default.

2. Review and, optionally, update the repository settings that the previous command has created in the **/etc/yum.repos.d/<repository_URL>.repo** file:

```
# cat /etc/yum.repos.d/<repository_URL>.repo
```

**WARNING**

Obtaining and installing software packages from unverified or untrusted sources other than Red Hat certificate-based **Content Delivery Network** (CDN) is a potential security risk, and can lead to security, stability, compatibility, and maintainability issues.

## 10.3. ENABLING A DNF REPOSITORY

You can enable a **DNF** repository added to your system by using the **dnf config-manager** command.

**Procedure**

- Enable a repository:

```
# dnf config-manager --enable <repository_id>
```

## 10.4. DISABLING A DNF REPOSITORY

You can disable a **DNF** repository added to your system by using the **dnf config-manager** command.

**Procedure**

- Disable a repository:

```
# dnf config-manager --disable <repository_id>
```

# CHAPTER 11. MANAGING VERSIONS OF APPLICATION STREAM CONTENT

Content in the AppStream repository can be available in multiple versions, corresponding to module streams.

## 11.1. MODULAR DEPENDENCIES AND STREAM CHANGES

Traditionally, packages providing content depend on further packages, and usually specify the desired dependency versions. For packages contained in modules, this mechanism applies as well, but the grouping of packages and their particular versions into modules and streams provides further constraints. Additionally, module streams can declare dependencies on streams of other modules, independent of the packages contained and provided by them.

After any operations with packages or modules, the whole dependency tree of all underlying installed packages must satisfy all the conditions that the packages declare. Additionally, all module stream dependencies must be satisfied. For example, disabling a module stream can require disabling other module streams. No packages will be removed automatically.

Note that the following actions can cause subsequent automatic operations:

- Enabling a module stream can result in enabling further module streams.
- Installing a module stream profile or installing packages from a stream can result in enabling further module streams and installing further packages.
- Removing a package can result in removing further packages. If these packages were provided by modules, the module streams remain enabled in preparation for further installation, even if no packages from these streams are installed any more. This mirrors the behavior of an unused **DNF** repository.

## 11.2. INTERACTION OF MODULAR AND NON-MODULAR DEPENDENCIES

Modular dependencies are an additional layer on top of regular RPM dependencies. Modular dependencies behave similarly to hypothetical dependencies between repositories. This means that installing different packages requires resolution of both the RPM dependencies and the modular dependencies.

The system will always retain the module and stream choices, unless explicitly instructed to change them. A modular package will receive updates contained in the currently enabled stream of the module that provides this package, but will not upgrade to a version contained in a different stream.

## 11.3. RESETTING MODULE STREAMS

Resetting a module is an action that returns this module to its initial state - neither enabled nor disabled. If the module has a configured default stream, this stream becomes active as a result of resetting the module.

Resetting the module is useful, for example, if you want to only extract the RPM content from the module without keeping the module enabled. You can use the **dnf module reset** command after enabling the module and extracting its contents to reset this module to its initial state.

**Procedure**

- Reset the module state:

```
# dnf module reset <module-name>
```

The module is returned to the initial state. Information about an enabled stream and installed profiles is erased but no installed content is removed.

## 11.4. DISABLING ALL STREAMS OF A MODULE

Modules that have a default stream will always have one stream active. If you want to make the content from all module streams of the module inaccessible, you can disable the whole module.

**Prerequisites**

- You understand the concept of an *active module stream*.

**Procedure**

- Disable the module:

```
# dnf module disable <module-name>
```

The **dnf** command asks for confirmation and then disables the module with all its streams. All of the module streams become inactive. No installed content is removed.

## 11.5. SWITCHING TO A LATER STREAM

When you switch to a later module stream, all respective packages are replaced with their later versions.

IMPORTANT

Back up your data and follow migration instructions specific to the component.

**Prerequisites**

- The system is fully updated.

**Procedure**

1. Switch the installed component to the new version and select the module (component) and stream (version):

```
# dnf module switch-to <module:stream>
```

For example, to switch from the **nodejs:18** module stream to the **nodejs:20** stream, enter:

```
# dnf module switch-to nodejs:20
...
Dependencies resolved.
================================================================================
=============
```

```
 Package          Arch   Version                                Repository      Size
================================================================================
=============
Upgrading:
 nodejs           x86_64 1:20.5.1-1.module+el9.3.0+19646+9a702805  rhel-AppStream  14 M
 nodejs-docs      noarch 1:20.5.1-1.module+el9.3.0+19646+9a702805  rhel-AppStream 8.0 M
 nodejs-full-i18n x86_64 1:20.5.1-1.module+el9.3.0+19646+9a702805  rhel-AppStream 8.5 M
 npm              x86_64 1:9.8.0-1.20.5.1.1.module+el9.3.0+19646+9a702805
                                                                   rhel-AppStream 2.6 M
Switching module streams:
 nodejs                  18 -> 20
```

You can also switch from non-modular content to a module stream. For example, to switch from non-modular **PHP 8.0** to modular **PHP 8.1**, enter:

```
# dnf module switch-to php:8.1
...
Dependencies resolved.
================================================================================
=============
 Package      Arch     Version                              Repository       Size
================================================================================
=============
Upgrading:
 php-common   x86_64   8.1.14-1.module+el9.2.0+17911+b059dfc2    rhel-AppStream   687 k
Enabling module streams:
 php                  8.1
```

2. Optional: Switch the installed component to the new version and select also the profile to be installed or updated:

```
# dnf module switch-to <module:stream/profile>
```

### Verification

- Verify that the installed component was switched to the new version (**[e]**):

```
$ dnf module list nodejs
...
rhel-AppStream
Name       Stream     Profiles                                 Summary
nodejs     18         common [d], development, minimal, s2i        Javascript runtime
nodejs     20 [e]     common [d] [i], development, minimal, s2i    Javascript runtime

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

# APPENDIX A. DNF COMMANDS LIST

In the following sections, examine **DNF** commands for listing, installing, and removing content in Red Hat Enterprise Linux 9.

## A.1. COMMANDS FOR LISTING CONTENT IN RHEL 9

The following are the commonly used DNF commands for finding content and its details in Red Hat Enterprise Linux 9:

| Command | Description |
|---|---|
| **dnf search *term*** | Search for a package by using term related to the package. |
| **dnf repoquery *package*** | Search for enabled **DNF** repositories for a selected package and its version. |
| **dnf list** | List information about all installed and available packages. |
| **dnf list --installed**<br>**dnf repoquery --installed** | List all packages installed on your system. |
| **dnf list --available**<br>**dnf repoquery** | List all packages in all enabled repositories that are available to install. |
| **dnf repolist** | List all enabled repositories on your system. |
| **dnf repolist --disabled** | List all disabled repositories on your system. |
| **dnf repolist --all** | List both enabled and disabled repositories. |
| **dnf repoinfo** | List additional information about the repositories. |
| **dnf info *package_name***<br>**dnf repoquery --info *package_name*** | Display details of an available package. |
| **dnf repoquery --info --installed *package_name*** | Display details of a package installed on your system. |
| **dnf module list** | List modules and their current status. |
| **dnf module info *module_name*** | Display details of a module. |
| **dnf module list *module_name*** | Display the current status of a module. |

| Command | Description |
|---|---|
| **dnf module info --profile *module_name*** | Display packages associated with available profiles of a selected module. |
| **dnf module info --profile *module_name:stream*** | Display packages associated with available profiles of a module by using a specified stream. |
| **dnf module provides *package*** | Determine which modules, streams, and profiles provide a package.<br><br>Note that if the package is available outside any modules, the output of this command is empty. |
| **dnf group summary** | View the number of installed and available groups. |
| **dnf group list** | List all installed and available groups. |
| **dnf group info *group_name*** | List mandatory and optional packages included in a particular group. |

## A.2. COMMANDS FOR INSTALLING CONTENT IN RHEL 9

The following are the commonly used **DNF** commands for installing content in Red Hat Enterprise Linux 9:

| Command | Description |
|---|---|
| **dnf install *package_name*** | Install a package.<br><br>If the package is provided by a module stream, **dnf** resolves the required module stream and enables it automatically while installing this package. This also happens recursively for all package dependencies. If more module streams satisfy the requirement, the default ones are used. |
| **dnf install *package_name_1 package_name_2*** | Install multiple packages and their dependencies simultaneously. |
| **dnf install *package_name.arch*** | Specify the architecture of the package by appending it to the package name when installing packages on a *multilib* system (AMD64, Intel 64 machine). |
| **dnf install */usr/sbin/binary_file*** | Install a binary by using the path to the binary as an argument. |

| Command | Description |
|---|---|
| **dnf install *<path>*** | Install a previously downloaded package from a local directory. |
| **dnf install *package_url*** | Install a remote package by using a package URL. |
| **dnf module enable *module_name:stream*** | Enable a module by using a specific stream.<br><br>Note that running this command does not install any RPM packages. |
| **dnf module install *module_name:stream***<br><br>**dnf install @*module_name:stream*** | Install a default profile from a specific module stream.<br><br>Note that running this command also enables the specified stream. |
| **dnf module install *module_name:stream/profile***<br><br>**dnf install @*module_name:stream/profile*** | Install a selected profile by using a specific stream. |
| **dnf group install *group_name*** | Install a package group by a group name. |
| **dnf group install *group_ID*** | Install a package group by the groupID. |

## A.3. COMMANDS FOR REMOVING CONTENT IN RHEL 9

The following are the commonly used **DNF** commands for removing content in Red Hat Enterprise Linux 9:

| Command | Description |
|---|---|
| **dnf remove *package_name*** | Remove a particular package and all dependent packages. |
| **dnf remove *package_name_1 package_name_2*** | Remove multiple packages and their unused dependencies simultaneously. |
| **dnf group remove group_name** | Remove a package group by the group name. |
| **dnf group remove *group_ID*** | Remove a package group by the groupID. |
| **dnf module remove --all *module_name:stream*** | Remove all packages from the specified stream.<br><br>Note that running this command can remove critical packages from your system. |

| Command | Description |
| --- | --- |
| **dnf module remove *module_name:stream/profile*** | Remove packages from an installed profile. |
| **dnf module remove *module_name:stream*** | Remove packages from all installed profiles within the specified stream. |
| **dnf module reset *module_name*** | Reset a module to the initial state.<br><br>Note that running this command does not remove packages from the specified module. |
| **dnf module disable *module_name*** | Disable a module and all its streams.<br><br>Note that running this command does not remove packages from the specified module. |