Red Hat Enterprise Linux 9

Managing software with the DNF tool

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

**MAKING OPEN SOURCE MORE INCLUSIVE** ................................................................. 4  

**PROVIDING FEEDBACK ON RED HAT DOCUMENTATION** ........................................ 5  

**CHAPTER 1. SOFTWARE MANAGEMENT TOOLS IN RED HAT ENTERPRISE LINUX 9** ............ 6  

**CHAPTER 2. DISTRIBUTION OF CONTENT IN RHEL 9** ............................................. 7  

2.1. REPOSITORIES ........................................... 7  
2.2. APPLICATION STREAMS ................................... 7  
2.3. MODULES ............................................. 8  
2.4. MODULE STREAMS ....................................... 8  
2.5. MODULE PROFILES ....................................... 9  

**CHAPTER 3. CONFIGURING DNF** ................................................................. 10  

3.1. VIEWING THE CURRENT DNF CONFIGURATIONS .............................................. 10  
3.2. SETTING DNF MAIN OPTIONS .......................................................... 10  
3.3. USING DNF PLUG-INS .......................................................... 10  
3.3.1. Managing DNF plug-ins .......................................................... 10  
3.3.2. Enabling and disabling DNF plug-ins ...................................................... 10  

**CHAPTER 4. SEARCHING FOR RHEL 9 CONTENT** ............................................... 12  

4.1. SEARCHING FOR SOFTWARE PACKAGES ..................................................... 12  
4.2. LISTING SOFTWARE PACKAGES .......................................................... 13  
4.3. LISTING REPOSITORIES .......................................................... 13  
4.4. DISPLAYING PACKAGE INFORMATION .................................................... 14  
4.5. LISTING PACKAGE GROUPS .......................................................... 14  
4.6. LISTING AVAILABLE MODULES AND THEIR CONTENTS ..................................... 15  
4.7. SPECIFYING GLOBAL Expressions IN DNF INPUT .......................................... 18  
4.8. ADDITIONAL RESOURCES .......................................................... 18  

**CHAPTER 5. INSTALLING RHEL 9 CONTENT** .................................................... 19  

5.1. INSTALLING PACKAGES .......................................................... 19  
5.2. INSTALLING PACKAGE GROUPS ..................................................... 20  
5.3. INSTALLING MODULAR CONTENT ................................................... 20  
5.4. RUNNING INSTALLED CONTENT ..................................................... 22  
5.5. CONFIGURING CUSTOM DEFAULT MODULE STREAMS AND PROFILES ......................... 23  
5.6. ADDITIONAL RESOURCES .......................................................... 24  

**CHAPTER 6. UPDATING RHEL 9 CONTENT** .................................................... 25  

6.1. CHECKING FOR UPDATES .......................................................... 25  
6.2. UPDATING PACKAGES .......................................................... 25  
6.3. UPDATING SECURITY-RELATED PACKAGES ...................................................... 26  

**CHAPTER 7. AUTOMATING SOFTWARE UPDATES IN RHEL 9** ................................... 27  

7.1. INSTALLING DNF AUTOMATIC ..................................................... 27  
7.2. DNF AUTOMATIC CONFIGURATION FILE ................................................... 27  
7.3. ENABLING DNF AUTOMATIC .......................................................... 28  
7.4. OVERVIEW OF THE SYSTEMD TIMER UNITS INCLUDED IN THE DNF-AUTOMATIC PACKAGE ...................................................... 29  

**CHAPTER 8. REMOVING RHEL 9 CONTENT** .................................................... 31  

8.1. REMOVING INSTALLED PACKAGES ..................................................... 31  
8.2. REMOVING PACKAGE GROUPS ..................................................... 31  
8.3. REMOVING INSTALLED MODULAR CONTENT ................................................... 32  
8.3.1. Removing packages from an installed profile .................................................. 32
8.3.2. Removing all packages from a module stream
8.4. ADDITIONAL RESOURCES

CHAPTER 9. HANDLING PACKAGE MANAGEMENT HISTORY ........................................... 39
  9.1. LISTING TRANSACTIONS 39
  9.2. REVERTING DNF TRANSACTIONS 40
    9.2.1. Reverting a single DNF transaction by using dnf history undo 40
    9.2.2. Reverting multiple DNF transactions by using dnf history rollback 42
  9.3. REPEATING TRANSACTIONS 43

CHAPTER 10. MANAGING CUSTOM SOFTWARE REPOSITORIES ................................. 44
  10.1. SETTING DNF REPOSITORY OPTIONS 44
  10.2. ADDING A DNF REPOSITORY 44
  10.3. ENABLING A DNF REPOSITORY 45
  10.4. DISABLING A DNF REPOSITORY 45

CHAPTER 11. MANAGING VERSIONS OF APPLICATION STREAM CONTENT .................. 47
  11.1. MODULAR DEPENDENCIES AND STREAM CHANGES 47
  11.2. INTERACTION OF MODULAR AND NON-MODULAR DEPENDENCIES 47
  11.3. resetting MODULE STREAMS 48
  11.4. DISABLING ALL STREAMS OF A MODULE 48

APPENDIX A. DNF COMMANDS LIST ................................................................. 49
  A.1. COMMANDS FOR LISTING CONTENT IN RHEL 9 49
  A.2. COMMANDS FOR INSTALLING CONTENT IN RHEL 9 50
  A.3. COMMANDS FOR REMOVING CONTENT IN RHEL 9 51
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Submitting comments on specific passages

1. View the documentation in the Multi-page HTML format and ensure that you see the Feedback button in the upper right corner after the page fully loads.

2. Use your cursor to highlight the part of the text that you want to comment on.

3. Click the Add Feedback button that appears near the highlighted text.

4. Add your feedback and click Submit.

Submitting feedback through Bugzilla (account required)

1. Log in to the Bugzilla website.

2. Select the correct version from the Version menu.

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 Submit Bug.
In Red Hat Enterprise Linux 9, software installation is ensured by the DNF tool. Red Hat continues to support the usage of the yum term for consistency with previous major versions of RHEL. If you type yum instead of dnf, the command works as expected because both are aliases for compatibility.

**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:

- Content distribution between repositories.
- Application Streams.
- Modules.
- Module streams.
- Module profiles.

2.1. REPOSITORIES

Red Hat Enterprise Linux 9 content is distributed through the two main repositories: BaseOS and AppStream. Both the BaseOS and AppStream content sets are required for a basic RHEL installation and are available with all RHEL subscriptions. For installation instructions, see the Performing a standard RHEL 9 installation document.

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 Red Hat Enterprise Linux.

AppStream

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

CodeReady Linux Builder

The CodeReady Linux Builder repository is available with all RHEL subscriptions. It provides additional packages for use by developers. Packages included in the CodeReady Linux Builder repository are unsupported.

Additional resources

- Performing a standard RHEL 9 installation
- Package manifest

2.2. APPLICATION STREAMS

Multiple versions of user-space components are delivered as Application Streams and updated more frequently than the core operating system packages. This provides greater flexibility to customize RHEL without impacting the underlying stability of the platform or specific deployments.

Each Application Stream component has a given lifecycle, either the same as RHEL 9 or shorter, more suitable to the particular application. For RHEL life cycle information, see Red Hat Enterprise Linux Life Cycle and Red Hat Enterprise Linux Application Streams Life Cycle.

Application Streams are available in the following formats:

- the familiar RPM format
as an extension to the RPM format called modules

as Software Collections

as Flatpaks.

RHEL 9 improves Application Streams experience by providing initial Application Stream versions that can be simply installed as RPM packages by using the traditional `dnf install` command.

**NOTE**

Certain initial Application Streams in the RPM format have a shorter life cycle than Red Hat Enterprise Linux 9.

Some additional Application Stream versions will be distributed as modules with a shorter lifecycle in future minor RHEL 9 releases.

Always determine what version of an Application Stream you want to install and make sure to review the [Red Hat Enterprise Linux Application Stream Lifecycle](#) first.

**NOTE**

Not all modules are Application Streams. Modular dependencies are not considered Application Streams.

Additional resources

- [Red Hat Enterprise Linux Life Cycle](#)
- [Red Hat Enterprise Linux Application Streams Life Cycle](#)
- [Red Hat Enterprise Linux 9: Application Compatibility Guide](#)
- Package manifest

## 2.3. MODULES

A module is a set of RPM packages that represent a component and are usually installed together. A typical module contains packages with an application, packages with the application-specific dependency libraries, packages with documentation for the application, and 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 represent versions of the AppStream components. Each of the streams receives updates independently.

Module streams can be active or inactive. Active streams give the system access to the RPM packages within the particular module stream, allowing installation of the respective component version. Streams are active if they are explicitly enabled by a user action.

Only one stream of a particular module can be active at a given point in time. Therefore, only one version of a component can be installed on a system. Different versions can be used in separate containers.
Each module can have a default stream. In RHEL 9, no default streams are defined but you can configure your own as described in Configuring custom default module streams and profiles.

Certain module streams can depend on other module streams.

To select 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
- Life cycle length 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 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 does not prevent installing or uninstalling any of the packages provided by the module. 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 when no profile is explicitly specified. However, 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:

- **common** - the production-ready packages.
  - This is the default profile.
- **development** - packages necessary for making modifications to Node.js.
- **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 information for DNF and related utilities is stored in the `/etc/dnf/dnf.conf` file. This file contains one mandatory `[main]` section, which enables you to set DNF options that have global effect.

In the following sections, learn how to configure DNF using the `/etc/dnf/dnf.conf` configuration file:

- **View the current DNF configurations.**
- **Set DNF `[main]` options.**
- **Use DNF plug-ins.**

3.1. VIEWING THE CURRENT DNF CONFIGURATIONS

To display the current DNF configuration, complete the following steps.

**Procedure**

- To display the current values of global DNF options specified in the `[main]` section of the `/etc/dnf/dnf.conf` file, use:

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

3.2. SETTING DNF MAIN OPTIONS

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

You can add additional options under the `[main]` section heading in `/etc/dnf/dnf.conf`.

For a complete list of available `[main]` options, see the `[main] OPTIONS` section of the `dnf.conf(5)` man page.

3.3. USING DNF PLUG-INS

DNF provides plug-ins that extend and enhance its operations. Certain plug-ins are installed by default.

3.3.1. Managing DNF plug-ins

The plug-in configuration files always contain a `[main]` section in which the `enabled=` option controls whether the plug-in is enabled when you run `dnf` commands. If this option is missing, you can add it manually to the file.

Every installed plug-in may have its own configuration file in the `/etc/dnf/plugins/` directory. You can enable or disable plug-in specific options in these files.

3.3.2. Enabling and disabling DNF plug-ins

In the DNF tool, plug-ins are loaded by default.

To modify loading of DNF plug-ins, and enable or disable specific DNF plug-ins, complete the following steps.
Procedure

- To disable or enable loading of DNF plug-ins, ensure a line beginning with `plugins=` is present in the [main] section of the `/etc/dnf/dnf.conf` file.

  1. To disable loading of DNF plug-ins, set the value of `plugins=` to 0.

    **IMPORTANT**
    
    Disabling all plug-ins is not advised. Certain plug-ins provide important DNF services and commands. In particular, the product-id and subscription-manager plug-ins provide support for the certificate-based Content Delivery Network (CDN). Disabling plug-ins globally is provided as a convenience option, and is advisable only when diagnosing a potential problem with DNF.

  2. To enable loading of DNF plug-ins, set the value of `plugins=` to 1.

- 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 the `update` command:

  ```
  # 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 certain DNF plug-ins for the `update` command:

  ```
  # dnf update --disableplugin=plugin-name
  ```

  Replace `plugin-name` with the name of the plug-in.

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

  ```
  # dnf update --enableplugin=plugin-name
  ```

  Replace `plugin-name` with the name of the plug-in.
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:

- Search for packages providing desired content.
- List installed and available packages.
- List enabled and disabled repositories.
- Display information about installed and available packages.
- List installed and available package groups.
- List available modules and find out details about them.
- Specify global expressions in dnf input.

4.1. SEARCHING FOR SOFTWARE PACKAGES

To find a package that provides a particular application or other content, complete the following steps.

Procedure

- To search for a package, use:
  
  ```
  $ dnf search term
  ```

  Replace term with a term related to the package.

  Note that the `dnf search` command returns term matches within the name and summary of the packages. This makes the search faster and you can search for packages you do not know the name of, but for which you know a related term.

- To include term matches within package descriptions, use:
  
  ```
  $ dnf search --all term
  ```

  Replace term with a term you want to search for in a package name, summary, or description.

  Note that the `dnf search --all` command enables a more exhaustive but slower search.

- To search for a package and its version or a file:
  
  ```
  $ dnf repoquery name
  ```

  Replace name with the name of a package or file you want to search for.

- To search for a package provide, binary, or file:
  
  ```
  $ dnf provides name
  ```

  Replace name with the name of a package provide, binary, or file you want to search for.
4.2. LISTING SOFTWARE PACKAGES

To list installed packages and packages that are available for installation, complete the following steps.

**Procedure**

- To list information about all installed and available packages, use:
  
  $ dnf list --all

- To list all packages installed on your system, use:
  
  $ dnf list --installed

  Alternatively:

  $ dnf repoquery --installed

- To list all packages in all enabled repositories that are available to install, use:
  
  $ dnf list --available

  Alternatively:

  $ dnf repoquery

**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 list enabled and disabled repositories on your system, complete the following steps.

**Procedure**

- To list all enabled repositories on your system, use:
  
  $ dnf repolist

- To list all disabled repositories on your system, use:
  
  $ dnf repolist --disabled

- To list both enabled and disabled repositories, use:
  
  $ dnf repolist --all

- To list additional information about the repositories, use:
4.4. DISPLAYING PACKAGE INFORMATION

To display information about available packages or packages installed on your system, complete the following steps.

Procedure

- To display information about one or more available packages, use:

  \$ dnf info package-name

  Replace \textit{package-name} with the name of the package.

  Alternatively:

  \$ dnf repoquery --info package-name

  Replace \textit{package-name} with the name of the package.

- To display information about one or more packages installed on your system, use:

  \$ dnf repoquery --info --installed package-name

  Replace \textit{package-name} with the name of the package.

**NOTE**

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

4.5. LISTING PACKAGE GROUPS

To list package groups installed on your system or package groups available for installation, complete the following steps.

Procedure

- To view the number of installed and available groups, use:

  \$ dnf group summary

- To list all installed and available groups, use:

  \$ dnf group list
Note that you can filter the results by appending command line options for the `dnf group list` command (`--hidden`, `--available`). For more available options see the man pages.

- To list mandatory and optional packages contained in a particular group, use:

  ```bash
  $ dnf group info group-name
  ```

  Replace `group-name` with the name of the group.

  **NOTE**

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

### 4.6. Listing Available Modules and Their Contents

To find which modules are available and what their details are, complete the following steps.

**Procedure**

- To list module streams available to your system:

  ```bash
  $ dnf module list
  ```

  The output of this command lists module streams with name, stream, profiles, and summary on a separate line.

- To display details about a module, including a description, a list of all profiles, and a list of all provided packages:

  ```bash
  $ dnf module info module-name
  ```

- To list which of these packages are installed by each of module profiles:

  ```bash
  $ dnf module info --profile module-name
  ```

- To display the current status of a module, including enabled streams and installed profiles:

  ```bash
  $ dnf module list module-name
  ```

- To find out which modules, streams, and profiles provide a specific package:

  ```bash
  $ dnf module provides package
  ```

  If the package is available outside any modules, the output of this command is empty.

**Example 4.1. Finding Out Details About a Module**

The following is an example of how to list available modules and how to obtain information about the `nodejs` module’s content.
NOTE

The outputs in this example have been edited for brevity. Actual outputs might contain more information than shown here.

1. List available modules:

```bash
$ dnf module list
(...)
rhel-AppStream

<table>
<thead>
<tr>
<th>Name</th>
<th>Stream</th>
<th>Profiles</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>maven</td>
<td>3.8</td>
<td>common [d]</td>
<td>Java project management and project comprehension tool</td>
</tr>
<tr>
<td>nodejs</td>
<td>18</td>
<td>common [d], Javascript runtime</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>development, minimal, s2i</td>
<td></td>
</tr>
<tr>
<td>php</td>
<td>8.1</td>
<td>common [d], PHP scripting language</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>devel, minimal</td>
<td></td>
</tr>
<tr>
<td>ruby</td>
<td>3.1</td>
<td>common [d]</td>
<td>An interpreter of object-oriented scripting language</td>
</tr>
</tbody>
</table>

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

2. Examine details of the nodejs module:

```bash
$ dnf module info nodejs

Name : nodejs
Stream : 18
Version : 9010020220808155010
Context : rhel9
Architecture : x86_64
Profiles : common [d], development, minimal, s2i
Default profiles : common
Repo : rhel-AppStream
Summary : Javascript runtime
Description : (...

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

If you do not specify any stream, `dnf` lists all available streams.

3. Examine profiles available in stream 18 of the nodejs module:

```bash
$ dnf module info --profile nodejs:18
(...)
Name : nodejs:18:9010020220808155010:rhel9:x86_64
common : nodejs
         : npm
development : nodejs
             : nodejs-devel
             : npm
minimal : nodejs
```
s2i : nodejs
   : nodejs-nodemon
   : npm

Note that each profile installs a different set of packages, including their dependencies.

4. Install the **nodejs:18** module stream with its **common** profile:

```bash
# dnf module install nodejs:18/common
(...)
```

```
Package      Architecture  Version
Repository
========================================================================

Installing group/module packages:
nodejs      x86_64        1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream   12 M
npm          x86_64        1:8.15.0-1.18.7.0.1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream   2.2 M
```

```
Installing weak dependencies:
nodejs-docs    noarch        1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream   7.2 M
nodejs-full-i18n x86_64        1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream   8.2 M
```

```
Installing module profiles:
nodejs/common
```

```
Enabling module streams:
nodejs                         18
```

```
Transaction Summary
========================================================================
Install  4 Packages
Total download size: 30 M
Installed size: 155 M
Is this ok [y/N]: y
(...)
```

5. Inspect the current status of the **nodejs** module:

```bash
$ dnf module list nodejs
(...)
```

```
Name   Stream Profiles       Summary
nodejs 18 [e] common [d] [i], Javascript runtime
development, minimal,
s2i
```

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

The output shows that the **nodejs:18** stream is enabled, and its **common** profile is installed.
Additional resources

- Modules
- Module streams
- Module profiles

4.7. SPECIFYING GLOBAL EXPRESSIONS IN DNF INPUT

With dnf commands, you can filter the results by appending one or more global expressions as arguments. Global expressions must be escaped when passed as arguments to the dnf command.

Procedure

- To ensure global expressions are passed to dnf as intended, use one of the following methods:
  - Enclose the entire global expression in single or double quotation marks:
    
    # dnf provides "/*/file-name"
    
    Replace file-name with the name of the file.
    
    Note that the file-name must be preceded either by / or */ character sequence to provide the desired outcome.
  - Escape the wildcard characters by preceding them with a backslash (\) character:
    
    # dnf provides "/\file-name"
    
    Replace file-name with the name of the file.

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:

- Install a package, package group, or sets of packages provided by modules, streams, and profiles by using the `dnf` command.
- Run RHEL 9 installed content.
- Configure custom default module streams and profiles.

5.1. INSTALLING PACKAGES

To install packages by using the `dnf install` command, complete the following steps.

Procedure

- To install a single package, use:

  ```
  # dnf install package-name
  ```

  Replace `package-name` with the name of the package.

  - If the package is not provided by any module stream, this procedure is identical to the procedure used on earlier versions of Red Hat Enterprise Linux.
  - If the package is provided by a module stream that is enabled, the package is installed without any further manipulation.
  - If the package is provided by a module stream that is not enabled, you must manually enable the respective module stream before installing the package.

- To install multiple packages and their dependencies simultaneously, use:

  ```
  # dnf install package-name-1 package-name-2
  ```

  Replace `package-name-1` and `package-name-2` with the names of the packages.

- When installing packages on a `multilib` system (AMD64, Intel 64 machine), you can specify the architecture of the package by appending it to the package name:

  ```
  # dnf install package-name.arch
  ```

  Replace `package-name.arch` with the name and architecture of the package.

- If you know the name of the binary you want to install, but not the package name, you can use the path to the binary as an argument:

  ```
  # dnf install /usr/sbin/binary-file
  ```

  Replace `/usr/sbin/binary-file` with a path to the binary file.

  `dnf` searches through the package lists, finds the package that provides `/usr/sbin/binary-file`, and prompts you as to whether you want to install it.
To install a previously-downloaded package from a local directory, use:

```bash
# dnf install /path/
```

Replace `/path/` with the path to the package.

**Additional resources**

- Installing modular content

## 5.2. INSTALLING PACKAGE GROUPS

To install a package group by a group name or by a groupID by using the `dnf group install` command, complete the following steps.

**Procedure**

- To install a package group by a group name, use:

  ```bash
  # dnf group install group-name
  ```

  Replace `group-name` with the full name of the group or environmental group.

- To install a package group by a groupID, use:

  ```bash
  # dnf group install groupID
  ```

  Replace `groupID` with the ID of the group.

## 5.3. INSTALLING MODULAR CONTENT

In RHEL 9, no default module streams are predefined. You must enable the required stream before installing a package.

To install modular content provided by a module stream or a profile by using the `dnf module install` command, complete the following steps.

**Prerequisites**

- You do not have any packages installed from another stream of the same module.

**Procedure**

- To install a selected module stream, use:

  ```bash
  # dnf module install module-name:stream
  ```

  The selected stream is automatically enabled. If a default profile is defined for the stream, this profile is automatically installed.

  Alternatively:

  1. Enable a module stream:
# dnf module enable module-name:stream

Replace `module-name` and `stream` with names of the module and stream. The module stream is now enabled but no packages are installed.

2. Install an active module stream that you have enabled:

```
# dnf module install module-name
```

- To install a selected profile of the module stream, use:

```
# dnf module install module-name:stream/profile
```

This enables the stream and installs the recommended set of packages for a given stream (version) and profile (purpose) of the module.

Example 5.1. Installing a stream of an application

The following is an example of how to install an application from a specific stream, namely, `Node.js` in version 18.

1. List modules that provide the `nodejs` package to see what streams are available:

```
$ dnf module list nodejs
(...)
Name        Stream  Profiles          Summary
nodejs      18      common [d], development,
               minimal,
               s2i
```

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

The output shows that the `nodejs` module is available with stream 18.

2. Install the packages provided by the `nodejs` module in stream 18:

```
# dnf module install nodejs:18
(...)
```

Dependencies resolved.

```
Package           Architecture        Version
Repository           Size
Installing group/module packages:
nodejs            x86_64              1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream    12 M
npm               x86_64              1:8.15.0-1.18.7.0.1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream    2.2 M
Installing weak dependencies:
nodejs-docs       noarch              1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream    7.2 M
```
nodejs-full-i18n  x86_64              1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
rhel-AppStream       8.2 M
Installing module profiles:
nodejs/common
Enabling module streams:
nodejs                                18

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

Install  4 Packages

Total download size: 30 M
Installed size: 155 M
Is this ok [y/N]: y
(…)
Complete!

Because the installation profile was not specified, the default profile `common` was used.

3. Verify the installed version of Node.js:

```
$ node -v
v18.7.0
```

Additional resources

- Modules
- Module streams
- Module profiles

5.4. RUNNING INSTALLED CONTENT

New commands are usually enabled after you install content from RHEL 9 repositories. If the commands originated from an RPM package or RPM packages were enabled by a module, the experience of using the command should be no different.

Procedure

- To run the new commands, enter them directly:

```
$ command
```

Replace `command` with the name of the command you want to run.
NOTE

In RHEL 9, GCC Toolset is packaged as a Software Collection. To run a command from a component packaged as a Software Collection, use:

```
$ scl enable collection 'command'
```

Replace `collection` with the name of the Software Collection.

For more information, see Using GCC Toolset.

5.5. CONFIGURING CUSTOM DEFAULT MODULE STREAMS AND PROFILES

In RHEL 9, no default streams are defined in the repository that contains the modules. You can configure a default stream and profile by creating a configuration file in the `/etc/dnf/modules.defaults.d/` directory.

When you define a custom default stream, you can install packages from this stream without enabling the stream first and without specifying the stream when installing the module.

When you define a custom default profile, this profile will be automatically chosen when you install the module stream for which it is defined.

To configure the default stream and profile through the `/etc/dnf/modules.defaults.d/` directory, complete the following steps.

Prerequisites

- You understand the concept of an active module stream.

Procedure

- Create a YAML configuration file in the `/etc/dnf/modules.defaults.d/` drop-in directory.

```yaml
document: modulemd-defaults
version: 1
data:
  module: <module>
  stream: "<stream>"
  profiles:
    '<stream>': [<profile>]
    '<stream>': [<profile>]
...
```

The preceding output represents the default definition present for the `<module>` module at the time of this writing. See the upstream specification of the modular metadata format.

Example 5.2. Configuring the `nodejs:18` module stream as the default stream and changing the default profile

The following is an example of how to configure the stream `18` of the `nodejs` module as the default stream and `minimal` profile as the default profile.
1. Examine the **nodejs** module:

```bash
# dnf module list nodejs
(...) rhel-AppStream
Name          Stream   Profiles          Summary
nodejs        18        common [d], Javascript runtime
               development,
               minimal,
               s2i

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

Currently, the **nodejs:18** module stream is not set as the default stream and the default profile is the **common** profile.

2. Configure the default stream to **18** and the default profile to **minimal** by implementing the following configuration in the YAML file in the `/etc/dnf/modules.defaults.d/` directory:

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

3. Examine the **nodejs** module again:

```bash
# dnf module list nodejs
(...) rhel-AppStream
Name          Stream   Profiles          Summary
nodejs        18 [d]   common, Javascript runtime
               development,
               minimal [d],
               s2i

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

The **nodejs:18** module stream is now set as a default stream and the **minimal** profile as the default profile of this stream.

5.6. ADDITIONAL RESOURCES

- **dnf(8)** man page
- Commands for installing content in RHEL 9
- Definition of the modulemd metadata YAML file
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, they are updated as well.

In the following sections, learn how to update content in Red Hat Enterprise Linux 9 by using DNF:

- Check the available updates.
- Update a single package, package group, or all packages and their dependencies.
- Apply security updates.

6.1. CHECKING FOR UPDATES

To check the available updates for packages installed on your system by using the `dnf check-update` command, complete the following steps.

Procedure

- Run the following command to see which packages installed on your system have available updates:

  ```
  # dnf check-update
  ```

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

6.2. UPDATING PACKAGES

To update a single package, a package group, or all packages and their dependencies by using the `dnf upgrade` command, complete the following steps.

Procedure

- To update all packages and their dependencies, use:

  ```
  # dnf upgrade
  ```

- To update a single package, use:

  ```
  # dnf upgrade package-name
  ```

  Replace `package-name` with the name of the package.

- To update a package group, use:

  ```
  # dnf group upgrade group-name
  ```

  Replace `group-name` with the name of the package group.
6.3. UPDATING SECURITY-RELATED PACKAGES

To update security-related packages by using the `dnf upgrade` or `dnf upgrade-minimal` command, complete the following steps.

**Procedure**

- To upgrade to the latest available packages that have security errata, use:

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

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

  ```bash
  # dnf upgrade-minimal --security
  ```
CHAPTER 7. AUTOMATING SOFTWARE UPDATES IN RHEL 9

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

**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.

In the following sections, learn how to automate software updates in Red Hat Enterprise Linux 9:

- Install the **DNF Automatic** tool.
- Enable the **DNF Automatic** tool.
- Examine the **DNF Automatic** configuration file and its sections.
- Examine the `dnf-automatic` systemd timer units.

### 7.1. INSTALLING DNF AUTOMATIC

To install the **DNF Automatic** tool on your system, complete the following steps.

**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]` section
  Sets the mode of operation of **DNF Automatic**.
• **[emitters]** section
  Defines how the results of **DNF Automatic** are reported.

• **[command_email]** section
  Provides the email emitter configuration for an external command used to send email.

• **[email]** section
  Provides the email emitter configuration.

• **[base]** section
  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 as standard output.

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`.

Additional resources

• DNF Automatic documentation

• `dnf-automatic(8)` man page

• [Overview of the systemd timer units included in the dnf-automatic package](#)

### 7.3. ENABLING DNF AUTOMATIC

To run **DNF Automatic**, you must always enable and start a specific systemd timer unit. You can use one of the timer units provided in the **dnf-automatic** package, or you can write your own timer unit depending on your needs.

To enable **DNF Automatic** on your system, complete the following steps.

**Prerequisites**

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

**Procedure**

• To select, enable, and start a systemd timer unit that downloads available updates, use:

  ```bash
  # systemctl enable dnf-automatic-download.timer
  # systemctl start dnf-automatic-download.timer
  ```

Red Hat Enterprise Linux 9 Managing software with the DNF tool
To select, enable, and start a systemd timer unit that downloads and installs available updates, use:

```
# systemctl enable dnf-automatic-install.timer
# systemctl start dnf-automatic-install.timer
```

To select, enable, and start a systemd timer unit that reports available updates, use:

```
# systemctl enable dnf-automatic-notifyonly.timer
# systemctl start dnf-automatic-notifyonly.timer
```

To select, enable, and start a systemd timer unit that downloads, downloads and installs, or reports available updates, use:

```
# systemctl enable dnf-automatic.timer
# systemctl start dnf-automatic.timer
```

Optionally, select, enable, and start a systemd timer unit in one command by using the `--now` option. For example:

```
# systemctl enable --now dnf-automatic-download.timer
```

**NOTE**

You can also run DNF Automatic by executing the `/usr/bin/dnf-automatic` file directly from the command line or from a custom script.

**Verification**

- Verify that the timer is enabled:

```
# systemctl status <systemd timer unit>
```

**Additional resources**

- dnf-automatic(8) man page
- Overview of the systemd timer units included in the dnf-automatic package
- DNF Automatic configuration file

### 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.

The `dnf-automatic` package includes the following systemd timer units:

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

<table>
<thead>
<tr>
<th>Timer unit</th>
<th>Function</th>
<th>Overrides settings in the <code>/etc/dnf/automatic.conf</code> file?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dnf-automatic-download.timer</code></td>
<td>Downloads packages to cache and makes them available for updating.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Note: This timer unit does not install the updated packages. To perform the installation, you must run the <code>dnf update</code> command.</td>
<td></td>
</tr>
<tr>
<td><code>dnf-automatic-install.timer</code></td>
<td>Downloads and installs updated packages.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>dnf-automatic-notifyonly.timer</code></td>
<td>Downloads only repository data to keep the repository cache up-to-date and notifies you about available updates.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Note: This timer unit does not download or install the updated packages.</td>
<td></td>
</tr>
<tr>
<td><code>dnf-automatic.timer</code></td>
<td>The behavior of this timer when downloading and applying updates is specified by the settings in the <code>/etc/dnf/automatic.conf</code> configuration file.</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Default behavior is the same as for the <code>dnf-automatic-download.timer</code> unit: it downloads packages, but does not install them.</td>
<td></td>
</tr>
</tbody>
</table>

**Additional resources**

- `dnf-automatic(8)` man page
- DNF Automatic configuration file
CHAPTER 8. REMOVING RHEL 9 CONTENT

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

- Remove a package.
- Remove a package group.
- Remove content installed from a module stream or a profile.
  - Remove packages from a selected profile.
  - Remove all packages from a selected module stream.

8.1. REMOVING INSTALLED PACKAGES

To remove packages installed on your system by using the dnf remove command, complete the following steps.

Procedure

- To remove a particular package and all unused dependent packages, use:
  
  ```bash
  # dnf remove package-name
  ```
  
  Replace package-name with the name of the package.

- To remove multiple packages and their unused dependencies simultaneously, use:
  
  ```bash
  # dnf remove package-name-1 package-name-2
  ```
  
  Replace package-name-1 and package-name-2 with the names of the packages.

**NOTE**

The dnf command removes a package together with any other dependent packages.

8.2. REMOVING PACKAGE GROUPS

To remove a package group either by the group name or the groupID by using the dnf group remove command, complete the following steps.

Procedure

- To remove a package group by the group name, use:
  
  ```bash
  # dnf group remove group-name
  ```
  
  Replace group-name with the full name of the group.

- To remove a package group by the groupID, use:
  
  ```bash
  # dnf group remove groupID
  ```
Replace groupID with the ID of the group.

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 will try 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 packages from a selected profile, complete the following steps.

Prerequisites

- The selected profile has been 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.
- You understand modular dependency resolution.

Procedure

1. Uninstall packages belonging to the selected profile:

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

   Replace `module-name`, `stream`, and `profile` with the module, stream, and profile you want to uninstall.

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

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

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

2. Check the list of packages under Removing: and Removing unused dependencies: before you proceed with the removal transaction.

To remove all packages from a selected stream, follow instructions in Section 8.3.2 "Removing all packages from a module stream".

Example 8.1. Removing packages from a selected profile
The following is an example of how to remove packages and their dependencies that belong to the development profile of the nodejs:18 module stream.

**NOTE**

The outputs in this example have been edited for brevity. Actual outputs might contain more information than shown here.

1. Install the nodejs:18 module stream, including all available profiles:

   ```
   # dnf module install nodejs:18/*
   (...) Dependencies resolved.
   Package Architecture Version
   Repository Size
   Installing group/module packages:
   nodejs x86_64 1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
   rhel-AppStream 12 M
   nodejs-devel x86_64 1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
   rhel-AppStream 202 k
   nodejs-nodemon noarch 2.0.15-1.module+el9.1.0+15718+e52ec601
   rhel-AppStream 762 k
   npm x86_64 1:8.15.0-1.18.7.0.1.module+el9.1.0+16284+4fdefb2f
   rhel-AppStream 2.2 M
   Installing dependencies:
   brotli x86_64 1.0.9-6.el9
   rhel-AppStream 314 k
   brotli-devel x86_64 1.0.9-6.el9
   rhel-AppStream 36 k
   ...
   Installing weak dependencies:
   nodejs-docs noarch 1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
   rhel-AppStream 7.2 M
   nodejs-full-i18n x86_64 1:18.7.0-1.module+el9.1.0+16284+4fdefb2f
   rhel-AppStream 8.2 M
   Installing module profiles:
   nodejs/common
   nodejs/development
   nodejs/minimal
   nodejs/s2i
   Transaction Summary
   Install 31 Packages
   Total download size: 36 M
   Installed size: 165 M
   Is this ok [y/N]: y
   (...) Complete!
   ```
2. Inspect the installed profiles:

   $ dnf module info nodejs

   Name             : nodejs
   Stream           : 18 [e] [a]
   Version          : 9010020221009220316
   Context          : rhel9
   Architecture     : x86_64
   Profiles         : common [d] [i], development [i], minimal [i], s2i [i]
   Default profiles : common
   Repo             : rhel-AppStream

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

   All profiles are installed as indicated in the output.

3. Remove packages from the **development** profile and their dependencies:

   # 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

4. Inspect the installed profiles after the removal:

   $ dnf module info nodejs

   Name             : nodejs
   Stream           : 18 [e] [a]
   Version          : 9010020221009220316
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 all packages from a selected module stream, complete the following steps.

**Prerequisites**

- The module stream has been enabled and at least some packages from the stream have been installed.
- You understand modular dependency resolution.

**Procedure**

1. Remove all packages from a selected stream:

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

   Replace `module-name` and `stream` with the module and stream you want to uninstall.

2. Check the list of packages under **Removing**: and **Removing unused dependencies**: before you proceed with the removal transaction.

3. Optionally, reset or disable the stream.

If you want to remove only packages from a selected profile, follow instructions in Section 8.3.1. "Removing packages from an installed profile".

**Example 8.2. Removing packages from the whole stream**

The following is an example of how to remove all packages from the `nodejs:18` module stream.

```bash
NOTE
The outputs in this example have been edited for brevity. Actual outputs might contain more information than shown here.
```

1. Install the `nodejs:18` module stream, including all available profiles:
# dnf module install nodejs:18/

(…)

Dependencies resolved.

========================================================================

Package          Architecture   Version

Repository       Size

========================================================================

Installing group/module packages:

nodejs           x86_64         1:18.10.0-3.module+el9.1.0+16866+0fab0697
    rhel-AppStream   13 M
nodejs-devel     x86_64         1:18.10.0-3.module+el9.1.0+16866+0fab0697
    rhel-AppStream   203 k
nodejs-nodemon   noarch         2.0.15-1.module+el9.1.0+15718+e52ec601
    rhel-AppStream   762 k
npm              x86_64         1:8.19.2-1.18.10.0.3.module+el9.1.0+16866+0fab0697
    rhel-AppStream   2.2 M

Installing dependencies:

brotli           x86_64         1.0.9-6.el9
    rhel-AppStream   314 k
brotli-devel     x86_64         1.0.9-6.el9
    rhel-AppStream   36 k

... 

Installing weak dependencies:

nodejs-docs      noarch         1:18.10.0-3.module+el9.1.0+16866+0fab0697
    rhel-AppStream   7.3 M
nodejs-full-i18n x86_64         1:18.10.0-3.module+el9.1.0+16866+0fab0697
    rhel-AppStream   8.2 M

Installing module profiles:

nodejs/common
nodejs/development
nodejs/minimal
nodejs/s2i

Enabling module streams:

nodejs                        18

Transaction Summary

========================================================================

Install  31 Packages

Total download size: 37 M
Installed size: 167 M
Is this ok [y/N]: y

2. Remove all packages from the nodejs:18 module stream:

# 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

3. Inspect the nodejs module after the removal:

```
$ 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
...
```

No Node.js module stream is currently installed.
Additional resources

- 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 or redo the transactions.

In the following sections, learn how to handle package management history by using the **dnf history** command:

- List the latest transactions, the latest operations for a selected package, and details of a particular transaction.
- Revert DNF transactions by using the **dnf history undo** and **dnf history rollback** commands:
  - Revert a single DNF transaction.
  - Revert all DNF transactions performed between the specified transaction and the last transaction.
- Repeat selected or last transactions.

9.1. LISTING TRANSACTIONS

To list the latest DNF transactions, the latest operations for a selected package, and details of a particular transaction, complete the following steps.

**Procedure**

- To display a list of all the latest DNF transactions, use:

  ```
  # dnf history
  ```

  **NOTE**

  The **Action(s)** column in the output 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.

- To display a list of all the latest operations for a selected package, use:

  ```
  # dnf history list package-name
  ```
Replace *package-name* with the name of the package.

- To display details of a particular transaction, use:

  ```sh
  # dnf history info transactionID
  ```

Replace *transactionID* with the ID of the transaction.

**NOTE**

You can filter the command output by appending global expressions. For more details, see Specifying global expressions in *dnf* input.

**Additional resources**

- [**dnf(8)**](#)

### 9.2. REVERTING DNF TRANSACTIONS

In RHEL 9, you can revert DNF transactions in two 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.

**IMPORTANT**

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

#### 9.2.1. Reverting a single DNF transaction by using dnf history undo

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 it. If the transaction uninstalled a package, *dnf history undo* installs it back. 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.

To revert a single transaction by using the *dnf history undo* command, complete the following steps.

**Procedure**

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

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

```bash
# dnf history info transaction_id
```

Replace `transaction_id` with the ID of a transaction you want to revert.

3. Revert the transaction:

```bash
# dnf history undo transaction_id
```

Replace `transaction_id` with the ID of a transaction you want to revert.

---

**Example 9.1. Reverting a single DNF transaction by using the `dnf history undo` command**

The following is an example of how to revert the installation transaction of the `unzip` package by using the `dnf history undo` command.

1. Install the `unzip` package:

```bash
# dnf install unzip
```

Dependencies resolved.

```
== Package Architecture Version Repository Size
=========================================================================
== Installing:
unzip x86_64 6.0-56.el9 rhel 186 k
```

Transaction Summary

```
==
Install 1 Package

Total download size: 186 k
Installed size: 392 k
Is this ok [y/N]: y
```

2. Identify the transaction ID that you want to revert:

```bash
# 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
```

3. Revert the transaction:

```bash
# dnf history undo 12
```
9.2.2. Reverting multiple DNF transactions by using `dnf history rollback`

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 will remain unchanged.

To revert multiple transactions by using the `dnf history rollback` command, complete the following steps.

**Procedure**

1. Identify the transaction ID of the state you want to revert to:
   
   ```
   # dnf history
   ```

2. Revert specified transactions:
   
   ```
   # dnf history rollback transaction_id
   ```

   Replace `transaction_id` with the transaction ID of the state you want to revert to.

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

   ```
   # dnf history rollback 1
   ```

**Example 9.2. Reverting multiple DNF transactions by using the `dnf history rollback` command**

The following is an example of how to roll back two installation transactions of the `unzip` and `zip` packages by using the `dnf history rollback` command.

1. Install the `unzip` package:

   ```
   # dnf install unzip
   Dependencies resolved.
   ==============================================================
   Package Architecture Version Repository Size
   ==============================================================
   Installing:
   unzip       x86_64     6.0-56.el9  rhel     186 k
   ``

   Transaction Summary
   ==============================================================
   Install 1 Package
   Total download size: 186 k
   Installed size: 392 k
   Is this ok [y/N]: y

2. Install the `wget` package:

   ```
   # dnf install wget
   ```
Dependencies resolved.

Package        Architecture      Version           Repository     Size
========================================================================
Installing:
wget           x86_64            1.21.1-7.el9      rhel           794 k

Transaction Summary
========================================================================
Install 1 Package

Total download size: 794 k
Installed size: 3.1 M
Is this ok [y/N]: y

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

```
# dnf history
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Command line</th>
<th>Date and time</th>
<th>Action(s)</th>
<th>Altered</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>install wget</td>
<td>2022-11-03 10:49</td>
<td>Install</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>install unzip</td>
<td>2022-11-03 10:49</td>
<td>Install</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>install vim-X11</td>
<td>2022-11-03 10:20</td>
<td>Install</td>
<td>171 EE</td>
</tr>
</tbody>
</table>

4. Revert the last two transactions:

```
# dnf history rollback 12
```

This action uninstalls the `wget` and `unzip` packages but leaves the `vim-X11` package installed.

9.3. REPEATING TRANSACTIONS

You can repeat steps performed during a transaction by using the `dnf history redo` command.

To repeat a selected transaction or the last DNF transaction, complete the following steps.

Procedure

- To repeat a particular transaction, use:

```
# dnf history redo transactionID
```

Replace `transactionID` with the ID of the transaction.

- To repeat the last transaction, use:

```
# dnf history redo last
```
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.

The configuration information for DNF and related utilities are stored in the `/etc/dnf/dnf.conf` file. This file contains the `[main]` section and can contain one or more `[repository]` sections, which allow you 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.

However, it is recommended to define individual repositories in new or existing `.repo` files in the `/etc/yum.repos.d/` directory.

In the following sections, learn how to manage custom software repositories by using DNF:

- Set the `[repository]` section options.
- Define a new DNF repository.
- Enable a DNF repository added to your system.
- Disable a DNF repository added to your system.

10.1. SETTING DNF REPOSITORY OPTIONS

The `/etc/dnf/dnf.conf` configuration file contains the `[repository]` sections, where `repository` is a unique repository ID. The `[repository]` sections allow you to define individual DNF repositories.

**NOTE**

Do not give custom repositories names used by the Red Hat repositories to avoid conflicts.

For a complete list of available `[repository]` options, see the `[repository] OPTIONS` section of the `dnf.conf(5)` man page.

10.2. ADDING A DNF REPOSITORY

To define a new repository, you can either:

- Add a `[repository]` section to the `/etc/dnf/dnf.conf` file.
- Add a `[repository]` section to a `.repo` file in the `/etc/yum.repos.d/` directory. Installed RPMs or software management tools, for example, Subscription Manager, can provide their own `.repo` file.

**NOTE**

Define your repositories in a `.repo` file instead of `/etc/dnf/dnf.conf` because all files with the `.repo` file extension in this directory are read by `dnf`. 
To add a DNF repository to your system by using the `dnf config-manager` command, complete the following steps.

**Procedure**

- Add a repository to your system:

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

  Replace `repository_url` with URL pointing to the repository.

**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

To enable a DNF repository added to your system by using the `dnf config-manager` command, complete the following steps.

**Procedure**

- Enable a repository:

  ```
  # dnf config-manager --enable repositoryID
  ```

  Replace `repositoryID` with the unique repository ID.

**Additional resources**

- [Listing repositories](#)

### 10.4. DISABLING A DNF REPOSITORY

To disable a DNF repository added to your system by using the `dnf config-manager` command, complete the following steps.

**Procedure**

- Disable a repository:

  ```
  # dnf config-manager --disable repositoryID
  ```

  Replace `repositoryID` with the unique repository ID.
Additional resources

Listing repositories
CHAPTER 11. MANAGING VERSIONS OF APPLICATION STREAM CONTENT

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

In the following sections, learn operations you must perform when changing existing enabled module streams:

- Modular dependency rules.
- Interaction of modular and non-modular dependencies.
- Reset modules to their initial state.
- Completely disable a module and all its 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.

As a result:

- Enabling a module stream can require enabling further module streams.
- Installing a module stream profile or installing packages from a stream can require enabling further module streams and installing further packages.
- Disabling a module stream can require disabling other module streams. No packages will be removed automatically.
- Removing a package can require 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 all of its streams to their 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.

To reset a module stream to its initial state by using the `dnf module reset` command, complete the following steps.

**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. In situations where the content from all the module streams must not be accessible, it is possible to disable the whole module.

To disable all streams of a module by using the `dnf module disable` command, complete the following steps.

**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.
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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dnf search term</code></td>
<td>Search for a package by using term related to the package.</td>
</tr>
<tr>
<td><code>dnf repoquery package</code></td>
<td>Search for enabled DNF repositories for a selected package and its version.</td>
</tr>
<tr>
<td><code>dnf list</code></td>
<td>List information about all installed and available packages.</td>
</tr>
<tr>
<td><code>dnf list --installed</code></td>
<td>List all packages installed on your system.</td>
</tr>
<tr>
<td><code>dnf repoquery --installed</code></td>
<td>List all packages installed on your system.</td>
</tr>
<tr>
<td><code>dnf list --available</code></td>
<td>List all packages in all enabled repositories that are available to install.</td>
</tr>
<tr>
<td><code>dnf repoquery</code></td>
<td>List all packages in all enabled repositories that are available to install.</td>
</tr>
<tr>
<td><code>dnf repolist</code></td>
<td>List all enabled repositories on your system.</td>
</tr>
<tr>
<td><code>dnf repolist --disabled</code></td>
<td>List all disabled repositories on your system.</td>
</tr>
<tr>
<td><code>dnf repolist --all</code></td>
<td>List both enabled and disabled repositories.</td>
</tr>
<tr>
<td><code>dnf repoinfo</code></td>
<td>List additional information about the repositories.</td>
</tr>
<tr>
<td><code>dnf info package-name</code></td>
<td>Display details of an available package.</td>
</tr>
<tr>
<td><code>dnf repoquery --info package_name</code></td>
<td>Display details of an available package.</td>
</tr>
<tr>
<td><code>dnf repoquery --info --installed package_name</code></td>
<td>Display details of a package installed on your system.</td>
</tr>
<tr>
<td><code>dnf module list</code></td>
<td>List modules and their current status.</td>
</tr>
<tr>
<td><code>dnf module info module-name</code></td>
<td>Display details of a module.</td>
</tr>
<tr>
<td><code>dnf module list module-name</code></td>
<td>Display the current status of a module.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>dnf module info --profile module-name</code></td>
<td>Display packages associated with available profiles of a selected module.</td>
</tr>
<tr>
<td><code>dnf module info --profile module-name:stream</code></td>
<td>Display packages associated with available profiles of a module by using a specified stream.</td>
</tr>
<tr>
<td><code>dnf module provides package</code></td>
<td>Determine which modules, streams, and profiles provide a package. Note that if the package is available outside any modules, the output of this command is empty.</td>
</tr>
<tr>
<td><code>dnf group summary</code></td>
<td>View the number of installed and available groups.</td>
</tr>
<tr>
<td><code>dnf group list</code></td>
<td>List all installed and available groups.</td>
</tr>
<tr>
<td><code>dnf group info group-name</code></td>
<td>List mandatory and optional packages included in a particular group.</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dnf install package-name</code></td>
<td>Install a package.</td>
</tr>
<tr>
<td></td>
<td>If the package is provided by a module stream, <code>dnf</code> 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.</td>
</tr>
<tr>
<td><code>dnf install package-name-1 package-name-2</code></td>
<td>Install multiple packages and their dependencies simultaneously.</td>
</tr>
<tr>
<td><code>dnf install package-name.arch</code></td>
<td>Specify the architecture of the package by appending it to the package name when installing packages on a multilib system (AMD64, Intel 64 machine).</td>
</tr>
<tr>
<td><code>dnf install /usr/sbin(binary-file)</code></td>
<td>Install a binary by using the path to the binary as an argument.</td>
</tr>
</tbody>
</table>
### 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dnf remove <em>package-name</em></td>
<td>Remove a particular package and all dependent packages.</td>
</tr>
<tr>
<td>dnf remove <em>package-name</em>1* <em>package-name</em>2</td>
<td>Remove multiple packages and their unused dependencies simultaneously.</td>
</tr>
<tr>
<td>dnf group remove <em>group-name</em></td>
<td>Remove a package group by the group name.</td>
</tr>
<tr>
<td>dnf group remove <em>groupID</em></td>
<td>Remove a package group by the groupID.</td>
</tr>
<tr>
<td>dnf module remove --all *module-name:*stream</td>
<td>Remove all packages from the specified stream. Note that running this command can remove critical packages from your system.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>dnf module remove module-name:stream/profile</code></td>
<td>Remove packages from an installed profile.</td>
</tr>
<tr>
<td><code>dnf module remove module-name:stream</code></td>
<td>Remove packages from all installed profiles within the specified stream.</td>
</tr>
</tbody>
</table>
| `dnf module reset module-name` | Reset a module to the initial state.  
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.  
Note that running this command does not remove packages from the specified module. |