Red Hat Enterprise Linux 8

Installing, managing, and removing user-space components

Managing content in the BaseOS and AppStream repositories by using the YUM software management tool

Last Updated: 2025-05-20

Red Hat Enterprise Linux 8 Installing, managing, and removing user-space components

Managing content in the BaseOS and AppStream repositories by using the YUM software management tool

Legal Notice

Abstract

Find, install, and utilize content distributed through the BaseOS and AppStream repositories by using the YUM 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 8

In Red Hat Enterprise Linux (RHEL) 8, use **YUM** to manage software. **YUM** is based on the **DNF** technology, which adds support for modular features.

NOTE

Upstream documentation identifies the technology as **DNF**, and the tool is referred to as **DNF**. As a result, some output returned by the new **YUM** tool in RHEL 8 mentions **DNF**.

Although **YUM** is based on **DNF**, it is compatible with **YUM** used in RHEL 7. For software installation, the **yum** command and most of its options work the same way in RHEL 8 as they did in RHEL 7.

Selected **YUM** plug-ins and utilities have been ported to the new **DNF** back end and can be installed under the same names as in RHEL 7. Packages also provide compatibility symlinks. Therefore, you can find binaries, configuration files, and directories in usual locations.

NOTE

The legacy Python API provided by **YUM** in RHEL 7 is no longer available. You can migrate your plug-ins and scripts to the new DNF Python API provided by **YUM** in RHEL 8. For more information, see DNF API Reference.

CHAPTER 2. DISTRIBUTION OF CONTENT IN RHEL 8

In the following sections, learn how the software is distributed in Red Hat Enterprise Linux 8.

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. Content in AppStream is available in the following formats:

- RPM packages
- Modules, which are an extension to the RPM format

IMPORTANT

Both the BaseOS and AppStream content sets are required by RHEL and are available in all RHEL subscriptions. For installation instructions, see Interactively installing RHEL from installation media.

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 Enterprise Linux (RHEL) 8 introduces the concept of multiple versions of user-space components named Application Streams. These components are delivered and 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 delivered through the AppStream repository.

Application Streams are available in the following formats:

- RPM packages
- Modules, which are an extension to the RPM format

IMPORTANT

Each Application Stream has its own life cycle, and it can be the same or shorter than the life cycle of RHEL 8.For more information, 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 Life Cycle
- Red Hat Enterprise Linux Application Streams Life Cycle

2.3. INTRODUCTION TO 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

2.3.1. 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. Default streams make it easy to consume RHEL packages the usual way without the need to learn about modules. The default stream is active unless the whole module has been disabled or another stream of that module has been enabled.

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.

IMPORTANT

The default stream does not change throughout the RHEL major release. Always consider each stream’s life cycle. Do not rely on the default stream for instances in which the default stream reaches the End of Life status prior to the end of the RHEL major release.

Certain module streams depend on other module streams. For example, the following module streams depend on certain **perl** module streams:

- **perl-App-cpanminus**
- **perl-DBD-MySQL**
- **perl-DBD-Pg**
- **perl-DBD-SQLite**
- **perl-DBI**
- **perl-YAML**
- **freeradius**

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 it.
- 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

2.3.2. 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 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 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. httpd module profiles

The **httpd** module, which provides the **Apache** web server, offers the following profiles for installation:

```
# yum module list httpd
```

```
Name          Stream          Profiles                          Summary
httpd         2.4 [d]         common [d], devel, minimal         Apache HTTP Server

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]**).
- **devel**: The packages necessary for making modifications to **httpd**.
- **minimal**: The smallest set of packages that provides a running **Apache** web server.

CHAPTER 3. CONFIGURING YUM

The configuration information for **YUM** and related utilities is stored in the **/etc/yum.conf** file. This file contains a mandatory **[main]** section that you can use to set **YUM** options that have global effect.

3.1. VIEWING THE CURRENT YUM CONFIGURATIONS

The **[main]** section in the **/etc/yum.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 **YUM** configuration:

```
# yum config-manager --dump
```

Additional resources

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

3.2. SETTING YUM MAIN OPTIONS

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

Procedure

1. Edit the **/etc/yum.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** sectiona in the **dnf.conf(5)** man page on you system

3.3. ENABLING AND DISABLING YUM PLUG-INS

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

WARNING

Disable all plug-ins only for diagnosing a potential problem. **YUM** 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 **YUM** uses plug-ins:
  - To enable or disable loading of **YUM** 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 **YUM** plug-ins.
    - Set **plugins=0** to disable loading of all **YUM** 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 **YUM** plug-ins for a particular command, append the **--noplugins** option to the command. For example, to disable **YUM** plug-ins for a single update command, enter:

    ```
    # yum --noplugins update
    ```

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

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

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

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

3.4. EXCLUDING PACKAGES FROM YUM OPERATIONS

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

NOTE

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

Procedure

- Exclude packages from the **YUM** operation by adding the following line to the **/etc/yum.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 yum input .

Additional resources

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

CHAPTER 4. SEARCHING FOR RHEL 8 CONTENT

In the following sections, learn how to locate and examine content in the AppStream and BaseOS repositories in Red Hat Enterprise Linux 8 by using **YUM**.

4.1. SEARCHING FOR SOFTWARE PACKAGES

To identify which package provides the software you require, you can use **YUM** 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:

    ```
    $ yum search <term>
    ```

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

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

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

4.2. LISTING SOFTWARE PACKAGES

You can use **YUM** to display a list of packages and their versions that are available in the repositories.

Procedure

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

  ```
  $ yum list --all
  ...
  cups.x86_64          1:2.2.6-57.el8      @rhel-AppStream
  cups-client.x86_64    1:2.2.6-57.el8      @rhel-AppStream
  cups-devel.i686       1:2.2.6-57.el8      rhel-AppStream
  cups-devel.x86_64     1:2.2.6-57.el8      rhel-AppStream
  ...
  ```

  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:

  ```
  $ yum list --all
  ...
  cups-1:2.2.6-57.el8.x86_64
  cups-client-1:2.2.6-57.el8.x86_64
  ```

```
cups-devel-1:2.2.6-57.el8.i686
cups-devel-1:2.2.6-57.el8.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 information, see Specifying global expressions in yum 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,:

   ```
   # yum 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:

   ```
   # yum repoinfo <repository_name>
   ```

   NOTE

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

4.4. DISPLAYING PACKAGE INFORMATION

You can query **YUM** 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:

  ```
  # yum info <package_name>
  ```

  This command displays the information for the currently installed package and, if available, its newer versions that are in the repository.

NOTE

You can filter the results by appending global expressions as arguments. For more information, see Specifying global expressions in yum 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:

   ```
   # yum group list
   ```

   Note that you can filter the results by appending the **--installed** and **--available** option to the **yum 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:

   ```
   # yum group info <group_name>
   ```

   NOTE

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

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

   ```
   # yum group summary
   ```

4.6. LISTING AVAILABLE MODULES AND THEIR CONTENTS

By searching for modules and displaying information about them with **YUM**, 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:

     ```
     $ yum module list
     Name        Stream   Profiles                              Summary
     ...
     nodejs       18       common [d], development, minimal, s2i   Javascript runtime
     postgresql   15       client, server [d]                    PostgreSQL server and client module
     ...

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

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

   - Search for which module provides a certain package:

     ```
     $ yum module provides <package_name>
     ```

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

     ```
     $ yum module provides npm
     ...
     npm-1:8.19.4-1.16.20.2.4.module+el8.9.0+21536+8fdee1fb.x86_64
     Module   : nodejs:16:8090020240315081818:a75119d5:x86_64
     Profiles : common development s2i
     Repo     : rhel-AppStream
     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:

     ```
     $ yum module info <module_name>
     ```

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

     ```
     $ yum module info nodejs
     ...
     Name             : nodejs
     Stream           : 20
     Version          : 8090020240228165436
     Context          : a75119d5
     Architecture     : x86_64
     Profiles         : common [d], development, minimal, s2i
     Default profiles : common
     Repo             : rhel-AppStream
     Summary          : Javascript runtime
     Description      : Node.js is a platform built on Chrome's JavaScript runtime for easily
     building fast, scalable network applications. Node.js uses an event-driven, non-blocking
     I/O model that makes it lightweight and efficient, perfect for data-intensive real-time
     applications that run across distributed devices.
     ```

```
Requires        : platform:[el8]
Artifacts       : nodejs-1:20.11.1-1.module+el8.9.0+21380+12032667.src
                : nodejs-1:20.11.1-1.module+el8.9.0+21380+12032667.x86_64
...
```

- List which packages each module profile installs:

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

NOTE

Each of the profiles installs a different set of packages, including their dependencies.

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

```
$ yum module info --profile nodejs
...
Name        : nodejs:18:8090020240301110609:a75119d5:x86_64
common      : nodejs
            : npm
development : nodejs
            : nodejs-devel
            : npm
minimal     : nodejs
s2i         : nodejs
            : nodejs-nodemon
            : npm
...
```

Additional resources

- Introduction to modules

4.7. SPECIFYING GLOBAL EXPRESSIONS IN YUM INPUT

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

Procedure

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

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

    Note that you must precede ***<file_name>*** either by a backslash (/) character 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:

    ```
    # yum provides \*/<file-name>
    ```

4.8. ADDITIONAL RESOURCES

- Commands for listing content in RHEL 8
- **yum(8)** man page on your system

CHAPTER 5. INSTALLING RHEL 8 CONTENT

In the following sections, learn how to install content in Red Hat Enterprise Linux 8 by using **YUM**.

5.1. INSTALLING PACKAGES

If a software is not part of the default installation, you can manually install it. **YUM** 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. For more information, see Enabling a module stream.

> **NOTE**
>
> If the package is provided by a module stream marked as default, **yum** automatically enables that module stream before installing this package.

Procedure

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

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

    If you install packages on a system that supports multiple architectures, such as AMD64 and Intel 64, you can specify the architecture of the package by appending it to the package name:

    ```
    # yum 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:

    ```
    # yum install <path_to_file>
    ```

  - To install a local RPM file, enter:

    ```
    # yum install <path_to_RPM_file>
    ```

    If the package has dependencies, specify the paths to these RPM files as well. Otherwise, **YUM** 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.

Procedure

- Use one of the following methods to install a package group:
  - To install a package group by a group name, enter one of the following commands::

    ```
    # yum group install <group_name>
    # yum install @<group_name>
    ```

  - To install a package group by the groupID, enter:

    ```
    # yum group install <group_ID>
    ```

5.3. ENABLING A MODULE STREAM

If the package you want to install is provided by a module stream, you must enable the respective module stream.

NOTE

If the package is provided by a module stream marked as default, **yum** automatically enables that module stream before installing this package.

IMPORTANT

It is recommended to always select a specific module stream for installation. Always consider each stream's life cycle.

Note that certain default module streams reach the End of Life status prior to the end of the RHEL major release.

Procedure

- Enable the module stream:

  ```
  # yum module enable <module_name>:<stream>
  ```

  NOTE

  If another stream of the module was previously active because it was default, it becomes inactive.

Additional resources

- Module streams
- Red Hat Enterprise Linux Application Streams Life Cycle

5.4. 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).

IMPORTANT

Always consider the module stream's life cycle.

Prerequisites

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

Procedure

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

```
# yum module list <module_name>
```

For example, to list modules that provide the **postgresql-server** package, enter:

```
$ yum module list postgresql
Name       Stream  Profiles          Summary
postgresql  9.6     client, server [d]  PostgreSQL server and client module
postgresql  10 [d]  client, server [d]  PostgreSQL server and client module
postgresql  12      client, server [d]  PostgreSQL server and client module
postgresql  13      client, server [d]  PostgreSQL server and client module
postgresql  15      client, server [d]  PostgreSQL server and client module

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

The output shows that the **postgresql** module is available with streams **9.6**, **10**, **12**, **13**, and **15**. The default stream is **10** (**[d]**).

2. Install a selected module stream:

```
# yum 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.

For example, to install the default profile (**server**) for stream **13** of the **postgresql** module, enter:

```
# yum module install postgresql:13
...
Dependencies resolved.
==========================================================================
=======================================
 Package              Architecture  Version                           Repository          Size
==========================================================================
=======================================
Installing group/module packages:
 postgresql-server       x86_64       13.10-1.module+el8.7.0+18279+1ca8cf12        rhel-
```

```
AppStream       5.6 M
Installing dependencies:
 libicu                x86_64       60.3-2.el8_1                              rhel               8.8 M
 libpq                 x86_64       13.5-1.el8                             rhel-AppStream       198 k
 postgresql                x86_64       13.10-1.module+el8.7.0+18279+1ca8cf12         rhel-
AppStream       1.5 M
Installing module profiles:
 postgresql/server
Enabling module streams:
 postgresql                         13
```

Verification

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

```
# yum module list postgresql
...
Name              Stream        Profiles                   Summary
postgresql        9.6          client, server [d]          PostgreSQL server and client module
postgresql        10 [d]        client, server [d]          PostgreSQL server and client module
postgresql        12           client, server [d]          PostgreSQL server and client module
postgresql        13 [e]        client, server [d] [i]      PostgreSQL server and client module
postgresql        15           client, server [d]          PostgreSQL server and client module
postgresql        16           client, server [d]          PostgreSQL server and client module

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

Additional resources

- Introduction to modules
- Red Hat Enterprise Linux Application Streams Life Cycle

5.5. SPECIFYING PACKAGE DETAILS FOR INSTALLATION

You can specify package details for a precise package installation process. To do so, append the following suffixes to the **yum install** command to explicitly define how to parse an argument:

- Use **-n** to specify the exact name of the package.
- Use **-na** to specify the exact package name and architecture.
- Use **-nevra** to specify the exact package name, epoch, version, release, and architecture.

Procedure

- Depending on your scenario, use one of the following options to optimize the package installation process:
  - To install a package by using its exact name, enter:

    ```
    # yum install-n <package_name>
    ```

  - To install a package by using its exact name and architecture, enter:

```
# yum install-na <package_name>.<architecture>
```

- To install a package by using its exact name, epoch, version, release, and architecture, enter:

```
# yum install-nevra <package_name>-<epoch>:<version>-<release>.<architecture>
```

5.6. ADDITIONAL RESOURCES

- Commands for installing content in RHEL 8
- **yum(8)** man page on your system

CHAPTER 6. UPDATING RHEL 8 CONTENT

With **YUM**, 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:

  ```
  # yum check-update
  ```

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

6.2. UPDATING PACKAGES

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

IMPORTANT

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

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

    ```
    # yum update
    ```

  - To update a single package, enter:

    ```
    # yum update <package_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 PACKAGE GROUPS

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

Procedure

- Update packages from a specific package group:

```
# yum group update <group_name>
```

IMPORTANT

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

6.4. UPDATING SECURITY-RELATED PACKAGES

You can use **YUM** to update packages that have security errata.

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:

    ```
    # yum update --security
    ```

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

    ```
    # yum update-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 8

DNF Automatic is an alternative command-line interface to **YUM** 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 available updates, 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:

  ```
  # yum 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]** section
  Provides 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 **YUM**.

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.

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 concerning downloading and applying updates.

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

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. This timer unit does not install the updated packages. To perform the installation, you must run the **yum update** command.	Yes
dnf-automatic-install.timer	Downloads and installs updated packages.	Yes
dnf-automatic-notifyonly.timer	Downloads only repository data to keep repository cache up-to-date and notifies you about available updates. 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** file. This timer downloads packages, but does not install them.	No

Additional resources

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

CHAPTER 8. REMOVING RHEL 8 CONTENT

In the following sections, learn how to remove content in Red Hat Enterprise Linux 8 by using **YUM**.

8.1. REMOVING INSTALLED PACKAGES

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

Procedure

- Remove particular packages:

```
# yum 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

- Use one of the following methods to install a package group:
  - To remove a package group by a group name, enter one of the following commands::

    ```
    # yum group remove <group_name>
    # yum remove @<group_name>
    ```

  - To remove a package group by the groupID, enter:

    ```
    # yum group remove <group_ID>
    ```

8.3. REMOVING INSTALLED MODULAR CONTENT

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

IMPORTANT

YUM 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 **yum module install *module_name*:*stream*/*profile*** command or as a default profile by using the **yum install *module_name*:*stream*** command.

Procedure

- Uninstall packages that belong to the selected profile:

  ```
  # yum module remove <module_name>:<stream>/<profile>
  ```

  For example, to remove packages from the **devel** profile of the **php:7.3** module stream, enter:

  ```
  # yum module remove php:7.3/devel
  (...)
  Dependencies resolved.
  ===============================================================================
   Package            Arch   Version                    Repository                Size
  ===============================================================================
  Removing:
   libzip           x86_64 1.5.2-1.module+el8.1.0+3189+a1bff096
                                          @rhel-8-for-x86_64-appstream-rpms 313 k
   php-devel          x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                          @rhel-8-for-x86_64-appstream-rpms 5.3 M
   php-pear          noarch 1:1.10.9-1.module+el8.1.0+3189+a1bff096
                                          @rhel-8-for-x86_64-appstream-rpms 2.1 M
   php-pecl-zip        x86_64 1.15.4-1.module+el8.1.0+3189+a1bff096
                                          @rhel-8-for-x86_64-appstream-rpms 119 k
   php-process         x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                          @rhel-8-for-x86_64-appstream-rpms 117 k
  Removing unused dependencies:
   autoconf          noarch 2.69-27.el8           @rhel-8-for-x86_64-appstream-rpms 2.2 M
  ...
  Disabling module profiles:
   php/devel

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

  Remove  64 Packages

  Freed space: 193 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:

```
# yum 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:

```
$ yum module info php
...
Name             : php
Stream           : 7.3 [e] [a]
Version          : 8020020200715124551
Context          : ceb1cf90
Architecture     : x86_64
Profiles         : common [d] [i], devel, minimal [i]
Default profiles : common
Repo             : rhel-AppStream
...
Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled, [a]ctive
```

All profiles except **devel** 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:

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

For example, to remove all packages from the **php:7.3** module stream, enter:

```
# yum module remove --all php:7.3
(...)
Dependencies resolved.
===================================================================================

 Package              Arch   Version                      Repository                    Size
===================================================================================

Removing:
 libzip               x86_64 1.5.2-1.module+el8.1.0+3189+a1bff096
                                                  @rhel-8-for-x86_64-appstream-rpms 313 k
 php-cli               x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms  11 M
 php-common             x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms 6.5 M
 php-devel             x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms 5.3 M
 php-fpm               x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms 5.6 M
 php-json              x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms  53 k
 php-mbstring           x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms 1.9 M
 php-pear              noarch 1:1.10.9-1.module+el8.1.0+3189+a1bff096
                                                  @rhel-8-for-x86_64-appstream-rpms 2.1 M
 php-pecl-zip           x86_64 1.15.4-1.module+el8.1.0+3189+a1bff096
                                                  @rhel-8-for-x86_64-appstream-rpms 119 k
 php-process            x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms 117 k
 php-xml               x86_64 7.3.5-5.module+el8.1.0+4560+e0eee7d6
                                                  @rhel-8-for-x86_64-appstream-rpms 557 k
Removing unused dependencies:
 autoconf             noarch 2.69-27.el8                @rhel-8-for-x86_64-appstream-rpms 2.2 M
...
Disabling module profiles:
 php/common
 php/devel
 php/minimal

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

Remove  73 Packages
```

```
Freed space: 220 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:

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

Verification

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

```
$ yum module info php
...
Name             : php
Stream           : 7.3 [e] [a]
Version          : 8020020200715124551
Context          : ceb1cf90
Architecture     : x86_64
Profiles         : common [d], devel, minimal
Default profiles : common
...
Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled, [a]ctive
```

The **7.3** stream of the **php** module is currently enabled (**[e]**) but no packages from this stream are installed (**[i]**).

Additional resources

- Modular dependencies and stream changes

8.4. SPECIFYING PACKAGE DETAILS FOR REMOVAL

You can specify package details for a precise package removal process. To do so, append the following suffixes to the **yum remove** command to explicitly define how to parse an argument:

- Use **-n** to specify the exact name of the package.
- Use **-na** to specify the exact package name and architecture.
- Use **-nevra** to specify the exact package name, epoch, version, release, and architecture.

Procedure

- Depending on your scenario, use one of the following options to optimize the package removal process:
  - To remove a package by using its exact name, enter:

    ```
    # yum remove-n <package_name>
    ```

  - To remove a package by using its exact name and architecture, enter:

    ```
    # yum remove-na <package_name>.<architecture>
    ```

  - To remove a package by using its exact name, epoch, version, release, and architecture, enter:

    ```
    # yum remove-nevra <package_name>-<epoch>:<version>-<release>.
    <architecture>
    ```

8.5. ADDITIONAL RESOURCES

- Commands for removing content in RHEL 8
- **yum(8)** man page on your system

CHAPTER 9. HANDLING PACKAGE MANAGEMENT HISTORY

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

- Timeline of **YUM** 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 **yum history** command to undo the transactions.

9.1. LISTING TRANSACTIONS

You can use **YUM** 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 **YUM** transactions, enter:

    ```
    # yum 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 **yum(8)** man page on your system.
  - To display a list of all the latest operations for a selected package, enter:

    ```
    # yum history list <package_name>
    ```
  - To display details of a particular transaction, enter:

    ```
    # yum history info <transaction_id>
    ```

NOTE

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

Additional resources

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

9.2. REVERTING YUM TRANSACTIONS

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

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

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

IMPORTANT

Downgrading RHEL system packages to an older version by using the **yum history undo** and **yum 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 8.1 to RHEL 8.0) is not recommended because it might leave the system in an incorrect state.

9.2.1. Reverting a single YUM transaction

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

- If the transaction installed a new package, **yum history undo** uninstalls the package.
- If the transaction uninstalled a package, **yum history undo** reinstalls the package.
- The **yum 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 **yum history undo** command fails.

Procedure

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

```
# yum 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:

```
# yum history info <transaction_id>
```

3. Revert the transaction:

```
# yum history undo <transaction_id>
```

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

```
# yum history undo 12
```

If you want to revert the last transaction, enter:

```
# yum history undo last
```

9.2.2. Reverting multiple YUM transactions

You can revert all **YUM** transactions performed between a specified transaction and the last transaction by using the **yum 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:

```
# yum 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:

```
# yum history rollback <transaction_id>
```

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

```
# yum history rollback 12
```

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

```
# yum history rollback 1
```

CHAPTER 10. MANAGING CUSTOM SOFTWARE REPOSITORIES

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

NOTE

It is recommended to define your repositories in the new or existing **.repo** file in **/etc/yum.repos.d/** because all files with the **.repo** file extension are read by YUM.

The **/etc/yum.conf** file contains the **[main]** sections 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/yum.conf** file override values set in the **[main]** section.

10.1. YUM REPOSITORY OPTIONS

The **/etc/yum.conf** configuration file contains the repository sections with a repository ID in brackets (**[*<repository_ID>*]**). You can use these sections to define individual YUM repositories.

IMPORTANT

Repository IDs 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 YUM REPOSITORY

You can add a YUM repository to your system by defining it in the **.repo** file in the **/etc/yum.repos.d/** directory.

Procedure

1. Add a repository to your system:

   ```
   # yum-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 YUM REPOSITORY

Once you added a **YUM** repository to your system, enable it to ensure installation and updates.

Procedure

- Enable a repository:

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

10.4. DISABLING A YUM REPOSITORY

To to prevent particular packages from installation or update, you can disable a **YUM** repository that contains these packages.

Procedure

- Disable a repository:

  ```
  # yum-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. For example, you can install a different version of a module that is already installed on your system.

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 **YUM** repository.

IMPORTANT

You cannot enable a stream of a module when another stream of the same module is already enabled. To switch streams, complete the steps in Switching to a later stream . Alternatively, reset the module, and then enable the new stream.

IMPORTANT

Removing all packages installed from a stream before switching to a different stream prevents the system from reaching states where packages could be installed with no repository or stream that provides them.

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 **yum module reset** command after enabling the module and extracting its contents to reset this module to its initial state.

Procedure

- Reset the module state:

  ```
  # yum 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:

  ```
  # yum module disable <module_name>
  ```

  Replace *module-name* with the name of the module that you want to disable.

  The **yum** 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.

Alternatively, you can remove all the module's content installed from the current stream, reset the module, and install the new stream.

Prerequisites

- The system is fully updated.
- No packages installed on the system are newer than the packages available in the repository.

Procedure

1. Determine if your system is prepared for switching to a later stream:

   ```
   # yum distro-sync
   ```

   IMPORTANT

   This command must finish with the **Nothing to do. Complete!**. If it instead proposes changes and asks for confirmation, carefully review these changes and consider whether you want to proceed. Run the **YUM distro-sync** command repeatedly, if necessary. Alternatively, you can refuse the suggested changes and manually modify your system to a state where the command returns **Nothing to do. Complete!**.

   By checking the **yum distro-sync** result before switching the streams, you prevent making changes to the system that are unrelated to the stream switching because the same command is required as the last step of this procedure.

2. Change the active stream to the later one:

   ```
   # yum module reset <module-name>
   # yum module enable <module-name>:<new-stream>
   ```

3. Synchronize installed packages to perform the change between streams:

   ```
   # yum distro-sync
   ```

   If this action suggests changes to content outside the streams, review them carefully.

NOTE

- If certain installed packages depend on the earlier stream, and there is no compatible version in the later stream, **YUM** reports a dependency conflict. In this case, use the **--allowerasing** option to remove such packages because they cannot be installed together with the later stream because of the missing dependencies.
- When switching **Perl** modules, you must always use the **--allowerasing** option because certain packages in the base RHEL 8 installation depend on **Perl 5.26**.
- You need to reinstall binary extensions for interpreted languages, which are typically written in C or C++, after the new stream is enabled. It concerns, for example, packages installed by using the following commands:
  - The **gem** command from the **ruby** module. For more information, see How to switch Ruby streams in RHEL 8 .
  - The **npm** command from the **nodejs** module
  - The **cpan** command from the **perl** module
  - The **pecl** command from the **php** module

11.6. DEFINING CUSTOM DEFAULT MODULE STREAMS AND PROFILES

By default, the **YUM** utility uses the default module streams defined in the repository that contains the modules. You can configure the default stream and default module profile in the **/etc/dnf/modules.defaults.d/** directory.

IMPORTANT

Always consider the module stream's life cycle.

Prerequisites

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

Procedure

1. Display the available streams and their profiles:

```
# yum module list <module_name>
```

For example, to list the available streams and their profiles of the **postgresql** module, enter:

```
# yum module list postgresql
(…)
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
Name          Stream     Profiles             Summary
postgresql     9.6       client, server [d]    PostgreSQL server and client module
postgresql     10 [d]     client, server [d]    PostgreSQL server and client module
postgresql     12        client, server [d]    PostgreSQL server and client module
postgresql     13        client, server [d]    PostgreSQL server and client module
postgresql      15          client, server [d]      PostgreSQL server and client module
…
Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

2. Create a **YAML** configuration file in the **/etc/dnf/modules.defaults.d/** drop-in directory.
   For example, create the **/etc/dnf/modules.defaults.d/postgresql.yaml** file with the following content to define **13** as the default stream and **server** as the default profile for the **postgresql** module:

   ```
   ---
   document: modulemd-defaults
   version: 1
   data:
     module: postgresql
     stream: "13"
     profiles:
       13: [server]
   ```

Verification

- Verify the default stream and profile settings:

  ```
  # yum module list postgresql
  (…)
  Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
  Name            Stream      Profiles              Summary
  postgresql      9.6         client, server        PostgreSQL server and client module
  postgresql      10          client, server        PostgreSQL server and client module
  postgresql      12          client, server        PostgreSQL server and client module
  postgresql      13 [d]      client, server [d]    PostgreSQL server and client module
  postgresql      15          client, server        PostgreSQL server and client module
  …
  Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
  ```

CHAPTER 12. YUM COMMANDS LIST

In the following sections, examine **YUM** commands for listing, installing, and removing content in Red Hat Enterprise Linux 8.

12.1. COMMANDS FOR LISTING CONTENT IN RHEL 8

The following are the commonly used YUM commands for finding content and its details in Red Hat Enterprise Linux 8:

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

Command	Description
yum module info --profile *module_name*	Display packages associated with available profiles of a selected module.
yum module info --profile *module_name:stream*	Display packages associated with available profiles of a module by using a specified stream.
yum module provides *package*	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.
yum group summary	View the number of installed and available groups.
yum group list	List all installed and available groups.
yum group info *group_name*	List mandatory and optional packages included in a particular group.

12.2. COMMANDS FOR INSTALLING CONTENT IN RHEL 8

The following are the commonly used **YUM** commands for installing content in Red Hat Enterprise Linux 8:

Command	Description
yum install *package_name*	Install a package. If the package is provided by a module stream, **yum** 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.
yum install *package_name_1 package_name_2*	Install multiple packages and their dependencies simultaneously.
yum 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).
yum install */usr/sbin/binary_file*	Install a binary by using the path to the binary as an argument.

Command	Description
yum install ***/path/***	Install a previously downloaded package from a local directory.
yum install ***package_url***	Install a remote package by using a package URL.
yum module enable ***module_name:stream***	Enable a module by using a specific stream. Note that running this command does not install any RPM packages.
yum module install ***module_name:stream*** **yum install** ***@module_name:stream***	Install a default profile from a specific module stream. Note that running this command also enables the specified stream.
yum module install ***module_name:stream/profile*** **yum install** ***@module_name:stream/profile***	Install a selected profile by using a specific stream.
yum group install ***group_name***	Install a package group by a group name.
yum group install ***group_ID***	Install a package group by the groupID.
yum install-n ***<package_name>***	Install a package by using its exact name.
yum install-na ***<package_name>.<architecture>***	Install a package by using its exact name and architecture.
yum install-nevra ***<package_name>-<epoch>:<version>-<release>.<architecture>***	Install a package by using its exact name, epoch, version, release, and architecture.

12.3. COMMANDS FOR REMOVING CONTENT IN RHEL 8

The following are the commonly used **YUM** commands for removing content in Red Hat Enterprise Linux 8:

Command	Description
yum remove ***package_name***	Remove a particular package and all dependent packages.
yum remove ***package_name_1 package_name_2***	Remove multiple packages and their unused dependencies simultaneously.
yum group remove group_name	Remove a package group by the group name.

Command	Description
yum group remove *group_ID*	Remove a package group by the groupID.
yum module remove --all *module_name:stream*	Remove all packages from the specified stream. Note that running this command can remove critical packages from your system.
yum module remove *module_name:stream/profile*	Remove packages from an installed profile.
yum module remove *module_name:stream*	Remove packages from all installed profiles within the specified stream.
yum module reset *module_name*	Reset a module to the initial state. Note that running this command does not remove packages from the specified module.
yum module disable *module_name*	Disable a module and all its streams. Note that running this command does not remove packages from the specified module.
yum remove-n *<package_name>*	Remove a package by using its exact name.
yum remove-na *<package_name>.<architecture>*	Remove a package by using its exact name and architecture.
yum remove-nevra *<package_name>-<epoch>:<version>-<release>.<architecture>*	Remove a package by using its exact name, epoch, version, release, and architecture.