Red Hat Enterprise Linux 9

使用 DNF 工具管理软件

使用 DNF 软件管理工具管理 RPM 存储库中的内容

Last Updated: 2024-10-12

Red Hat Enterprise Linux 9 使用 DNF 工具管理软件

使用 DNF 软件管理工具管理 RPM 存储库中的内容

法律通告

摘要

使用 DNF 工具查找、安装和使用通过 RPM 存储库分发的内容。了解如何与软件包、模块、流和配置文件一起工作。

目录

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 Jira 网站。
2. 点顶部导航栏中的 **Create**
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您对改进的建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 RED HAT ENTERPRISE LINUX 9 中的软件管理工具

在 Red Hat Enterprise Linux (RHEL) 9 中，使用 **DNF** 工具来管理软件。出于与之前的主 RHEL 版本的兼容性原因，您仍然可以使用 **yum** 命令。但是，在 RHEL 9 中， **yum** 是 **dnf** 的一个别名，它提供了与 **yum** 的一定程度的兼容性。

注意

虽然 RHEL 8 和 RHEL 9 基于 **DNF**，但它们与 RHEL 7 中使用的 **YUM** 兼容。

第 2 章 RHEL 9 发布的内容

在以下部分中，了解软件是如何在 Red Hat Enterprise Linux 9 中发布的。

2.1. 软件仓库

Red Hat Enterprise Linux (RHEL)通过不同的存储库分发内容，例如：

BaseOS

BaseOS 存储库中的内容由底层操作系统功能的核心组件组成，其为所有安装提供基础。此内容以 RPM 格式提供，它的支持条款与 RHEL 早期版本中的条款类似。

AppStream

AppStream 仓库的内容包括额外的用户空间应用程序、运行时语言和数据库来支持各种工作负载和使用案例。

重要

RHEL 需要 BaseOS 和 AppStream 内容集，它们在所有 RHEL 订阅中都提供。

CodeReady Linux Builder

CodeReady Linux Builder 存储库在所有 RHEL 订阅中可用。它为开发人员提供了额外的软件包。红帽不支持 CodeReady Linux Builder 存储库中包含的软件包。

其它资源

- 软件包清单

2.2. 应用程序流

红帽提供多个用户空间组件的版本作为应用程序流，它们比核心操作系统软件包的更新更频繁。这为自定义 Red Hat Enterprise Linux (RHEL)提供了更多的灵活性，而不影响底层平台或特定部署的稳定性。

提供以下格式的应用程序流：

- RPM 格式
- 模块，它们是 RPM 格式的扩展
- Software Collections

RHEL 9 通过提供初始应用程序流版本作为 RPM 来提高应用程序流体验，您可以使用 **dnf install** 命令安装它们。

从 RHEL 9.1 开始，红帽提供额外的应用程序流版本作为模块，其具有较短的生命周期。

重要

每个应用程序流都有自己的生命周期，它可能与 RHEL 9 的生命周期相同或更短。请参阅 Red Hat Enterprise Linux 应用程序流生命周期 。

始终决定您要安装应用程序流的哪个版本，并确保首先查看 RHEL 应用程序流生命周期。

其它资源

- Red Hat Enterprise Linux 9: 应用程序兼容性指南
- 软件包清单
- Red Hat Enterprise Linux Application Streams Life Cycle

2.3. 模块

模块是代表一个组件的一组 RPM 软件包。典型的模块包含以下软件包类型：

- 带有应用程序的软件包
- 带有特定于应用程序依赖项库的软件包
- 带有应用程序文档的软件包
- 带有助手工具的软件包

2.4. 模块流

模块流是可以作为 AppStream 物理存储库中的虚拟存储库的过滤器。AppStream 组件的模块流版本。每个流都独立接收更新，它们可以依赖于其他模块流。

模块流可以是活跃的或者不活跃的。活跃的流可让系统访问特定模块流中的 RPM 软件包，允许安装相应的组件版本。

在以下情况下，流处于活跃状态：

- 如果管理员明确启用了它。
- 如果流是启用的模块的依赖项。
- 如果流是默认流。每个模块都可以有一个默认流，但在 Red Hat Enterprise Linux 9 中，没有定义默认流。如果需要，您可以配置默认流，如 定义自定义默认模块流和配置文件 中所述。

给定时间点上只能激活一个特定模块的流。因此，只有特定流中的软件包可用。

在为运行时用户应用程序或开发人员应用程序选择特定流前，请考虑以下几点：

- 所需功能以及哪个组件版本支持该功能
- 与应用程序或用例的兼容
- 应用程序流的 生命周期 和您的更新计划

有关所有可用模块和流的列表，请查看软件包清单。有关每个组件的更改，请查看 发行注记。

其它资源

- 模块依赖关系和流更改

2.5. 模块配置集

模块配置文件是为特定用例要一起安装的推荐的软件包列表，如服务器、客户端、开发、最小安装或其他。这些软件包列表可以包含模块流以外的软件包，通常是来自 BaseOS 存储库或流依赖项的软件包。

使用配置集安装软件包是为方便用户提供的一次性操作。您还可以使用同一模块流的多个配置集安装软件包，而无需进一步准备步骤。

每个模块流可以有任何数量的配置集，包括没有。对于任何给定的模块流，其配置文件的某些部分可以被标记为 *default*，然后在您没有明确指定配置文件时用于配置文件安装操作。但是，模块流的默认配置文件的存在不是必需的。

例 2.1. nodejs 模块配置集

提供 Node.js 运行时环境的 **nodejs** 模块为安装提供以下配置集：

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

在本例中，提供了以下配置文件：

- **common** ：生产就绪软件包。这是默认的配置文件(**[d]**)。
- **development** ：生产就绪软件包，包括 Node.js 开发标头。
- **minimal** ：提供 Node.js 运行时环境的最小软件包集合。
- **s2i**: 创建 Node.js 源-到-镜像 (S2I) Linux 容器所需的软件包。

第 3 章 配置 DNF

DNF 及相关工具的配置存储在 **/etc/dnf/dnf.conf** 文件的 **[main]** 部分中。

3.1. 查看当前的 DNF 配置

/etc/dnf/dnf.conf 文件中的 **[main]** 部分仅包含已明确设置的设置。但是，您可以显示 **[main]** 部分的所有设置，包括尚未设置的部分，因此请使用它们的默认值。

流程

- 显示全局 DNF 配置：

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

其它资源

- 您系统上的 **dnf.conf (5)** 手册页

3.2. 设置 DNF 主选项

/etc/dnf/dnf.conf 文件包含一个 **[main]** 部分。本节中的键-值对会影响 DNF 如何操作及对待存储库。

流程

1. 编辑 **/etc/dnf/dnf.conf** 文件。
2. 根据您的要求更新 **[main]** 部分。
3. 保存更改。

其它资源

- **[main] OPTIONS** 和您系统上 **dnf.conf (5)** 手册页中的 **OPTIONS FOR BOTH [main] AND REPO** 部分。

3.3. 管理 DNF 插件

每个安装的插件都可以在 **/etc/dnf/plugins/** 目录中有自己的配置文件。在此目录中命名插件配置文件 ***<plug-in_name>*.conf**。默认情况下，插件通常是启用的。要在其中一个配置文件中禁用插件，请在文件中添加以下内容：

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

3.4. 启用和禁用 DNF 插件

在 DNF 工具中，插件会被默认加载。但是，您可以影响 DNF 加载哪个插件。

警告

禁用所有插件只用于诊断潜在的问题。**DNF** 需要某些插件，如 **product-id** 和 **subscription-manager**，禁用它们会导致 Red Hat Enterprise Linux 无法从 Content Delivery Network (CDN)安装或更新软件。

流程

- 使用以下方法之一会影响 **DNF** 如何使用插件：
  - 要在全局范围内启用或禁用 **DNF** 插件加载，请将 **plugins** 参数添加到 **/etc/dnf/dnf.conf** 文件的 **[main]** 部分中。
    - 设置 **plugins=1**（默认）以启用加载所有 **DNF** 插件。
    - 设置 **plugins=0** 以禁用加载所有 **DNF** 插件。
  - 要禁用特定的插件，请将 **enabled=False** 添加到 **/etc/dnf/plugins/*<plug-in_name>*.conf** 文件的 **[main]** 部分中。
  - 要禁用特定命令的所有 **DNF** 插件，请在命令中附加 **--noplugins** 选项。例如，要为单个 update 命令禁用 **DNF** 插件，请输入：

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

  - 要为单个命令禁用某些 **DNF** 插件，请在命令中附加 **--disableplugin=*plugin-name*** 选项。例如，要为单个 update 命令禁用某个 **DNF** 插件，请输入：

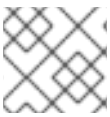
    ```
    # dnf update --disableplugin=<plugin_name>
    ```

  - 要为单个命令启用某些 **DNF** 插件，请在命令中附加 **--enableplugin=*plugin-name*** 选项。例如，要为单个 update 命令启用某个 **DNF** 插件，请输入：

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

3.5. 从 DNF 操作中排除软件包

您可以使用 **excludepkgs** 选项配置 **DNF**，来从任何 **DNF** 操作中排除软件包。您可以在 **[main]** 或 **/etc/dnf/dnf.conf** 文件的 repository 部分中定义 **excludepkgs**。



注意

您可以使用 **--disableexcludes** 选项，临时禁止从操作中排除配置的软件包。

流程

- 通过在 **/etc/dnf/dnf.conf** 文件中添加以下行，来从 **DNF** 操作中排除软件包：

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

或者，使用全局表达式而不是软件包名称来定义您要排除的软件包。如需更多信息，请参阅 在 DNF 输入中指定全局表达式。

其它资源

- 您系统上的 **dnf.conf (5)** 手册页
- 在 DNF 输入中指定全局表达式

第 4 章 搜索 RHEL 9 内容

在以下部分中，了解如何使用 DNF 查找和检查 Red Hat Enterprise Linux 9 中 AppStream 和 BaseOS 存储库中的内容。

4.1. 搜索软件包

要识别哪个软件包提供您需要的软件，您可以使用 DNF 搜索存储库。

流程

- 根据您的场景，使用以下选项之一来搜索存储库：
  - 要在软件包的名称或摘要中搜索术语，请输入：

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

  - 要在软件包的名称、概述或描述中搜索术语，请输入：

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

    请注意，在描述中使用 **--all** 选项进行额外搜索比普通搜索操作要慢。

  - 要搜索软件包名称，并在输出中列出软件包名称及其版本，请输入：

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

  - 要搜索哪个软件包提供了文件，请指定文件名或文件的路径：

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

4.2. 列出软件包

您可以使用 DNF 显示存储库中提供的软件包及其版本列表。如果需要，您可以过滤此列表，例如，只列出提供更新的软件包。

流程

- 列出所有可用软件包的最新版本，包括构架、版本号以及从其中安装的存储库：

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

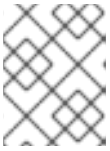
  存储库前面的 @ 符号表示此行中的软件包当前已安装。

  另外，要显示所有可用的软件包，包括版本号和架构，请输入：

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

另外，您可以使用其他选项而不是 **--all** 来过滤输出，例如：

- 使用 **--installed** 仅列出已安装的软件包。
- 使用 **--available** 列出所有可用软件包。
- 使用 **--upgrades** 列出新版本提供的软件包。



注意

您可以通过将全局表达式附加为参数来过滤结果。如需了解更多详细信息，请参阅 在 DNF 输入中指定全局表达式。

4.3. 列出软件仓库

要获得系统上启用和禁用的存储库的概述，您可以列出它们。

流程

1. 列出系统上所有启用的存储库：

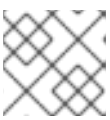
```
$ dnf repolist
```

要只显示某些存储库，请在命令中附加以下选项之一：

- 附加 **--disabled** 只列出禁用的存储库。
- 附加 **--all** 列出启用和禁用的存储库。

2. 可选：列出存储库的额外信息：

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



注意

您可以使用全局表达式过滤结果。详情请参阅 在 DNF 输入中指定全局表达式。

4.4. 显示软件包信息

您可以查询 DNF 存储库来显示软件包的详情，如下所示：

- 版本

- 发布
- 架构
- 软件包大小
- 描述

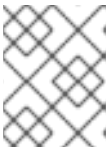
流程

- 显示一个或多个可用软件包的信息：

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

这个命令显示当前安装的软件包的信息，如果有的话，显示存储库中的较新版本。另外，使用以下命令显示存储库中具有指定名称的所有软件包的信息：

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



注意

您可以通过将全局表达式附加为参数来过滤结果。详情请参阅 在 DNF 输入中指定全局表达式。

4.5. 列出软件包组和它们提供的软件包

软件包组捆绑多个软件包，您可以使用软件包组在一个步骤中安装分配给组的所有软件包。但是，在安装前必须识别所需软件包组的名称。

流程

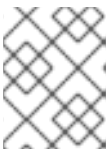
1. 列出已安装的和可用的组：

```
$ dnf group list
```

请注意，您可以通过在 **dnf group list** 命令中附加 **--installed** 和 **--available** 选项来过滤结果。通过使用 **--hidden** 选项，您可以在输出中显示隐藏的组。

2. 列出特定组中包含的强制的、可选的和默认的软件包：

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



注意

您可以通过将全局表达式附加为参数来过滤结果。如需了解更多详细信息，请参阅在 DNF 输入中指定全局表达式。

3. 可选：查看已安装和可用的组的数量：

```
$ dnf group summary
```

4.6. 列出可用模块及其内容

通过搜索模块并使用 DNF 显示有关它们的信息，您可以识别存储库中提供了哪些模块，并在安装模块前选择合适的流。

流程

1. 使用以下方法之一列出模块信息：

   - 列出所有可用模块：

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

     使用 **dnf module list *<module_name>*** 命令列出相同的信息，但只用于特定的模块。

   - 搜索哪个模块提供了某个软件包：

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

     例如，要显示哪些模块和配置文件提供了 **npm** 软件包，请输入：

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

2. 使用以下方法之一列出模块详情：

   - 列出模块的所有详情，包括描述、所有配置文件的列表以及模块提供的所有软件包的列表：

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

     例如，要显示 **nodejs** 软件包的详细信息，请输入：

     ```
     $ dnf module info nodejs
     Name             : nodejs
     Stream           : 18
     Version          : 9010020221009220316
     Context          : rhel9
     Architecture     : x86_64
     Profiles         : common [d], development, minimal, s2i
     Default profiles : common
     Repo             : rhel-9-for-x86_64-appstream-rpms
     Summary          : Javascript runtime
     Description      : Node.js is a platform built on Chrome's JavaScript runtime...
     Requires         : platform:[el9]
     ```

```
Artifacts       : nodejs-1:18.10.0-3.module+el9.1.0+16866+0fab0697.src
                : nodejs-1:18.10.0-3.module+el9.1.0+16866+0fab0697.x86_64
                : npm-1:8.19.2-1.18.10.0.3.module+el9.1.0+16866+0fab0697.x86_64
...
```

- 列出每个模块配置文件安装了哪些软件包：

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

例如，要显示 **nodejs** 模块的此信息，请输入：

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

其它资源

- 模块
- 模块流
- 模块配置集

4.7. 在 DNF 输入中指定全局表达式

您可以通过将一个或多个全局表达式附加为参数来过滤 **dnf** 命令的结果。

流程

- 如果您在 **dnf** 命令中使用全局表达式，请使用以下方法之一：
  - 将整个全局表达式用单引号或双引号括起来：

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

请注意，如果完整路径未知，您必需在 ***<file_name>*** 前加上/用于绝对路径或加上*/使用通配符。

  - 在它们前面使用反斜杠(\)字符转义通配符字符：

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

4.8. 其它资源

- 在 RHEL 9 中列出内容的命令

第 5 章 安装 RHEL 9 内容

在以下部分中，了解如何使用 **DNF** 在 Red Hat Enterprise Linux 9 中安装内容。

5.1. 安装软件包

如果软件不是默认安装的一部分，您可以手动安装它。**DNF** 会自动解析和安装依赖项。

先决条件

- 可选：您知道要安装的软件包的名称。
- 如果要安装的软件包由模块流提供，则会启用对应的模块流。

流程

- 使用以下方法之一安装软件包：
  - 要从存储库安装软件包，请输入：

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

    如果您在支持多个构架的系统上安装软件包，如 **i686** 和 **x86_64**，您可以通过将其附加到软件包名称来指定软件包的架构：

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

  - 如果您只知道软件包提供的文件的路径，但知道软件包名称，您可以使用此路径安装相应的软件包：

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

  - 要安装本地 RPM 文件，请输入：

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

    如果软件包有依赖项，还要指定到这些 RPM 文件的路径。否则，**DNF** 会从存储库下载依赖项，如果存储库中没有，则会失败。

其它资源

- 安装模块化内容

5.2. 安装软件包组

软件包组捆绑多个软件包，您可以使用软件包组在一个步骤中安装分配给组的所有软件包。

先决条件

- 您知道您要安装的组的名称或 ID。

流程

- 安装软件包组：

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

5.3. 安装模块化内容

对于某些软件，红帽提供了模块。您可以使用模块安装特定版本（流）和一组软件包（配置文件）。

流程

1. 列出提供您要安装的软件包的模块：

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

例如，要列出 **nodejs** 模块的详细信息，请输入：

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

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

2. 安装模块：

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

如果定义了流的默认配置文件，您可以在命令中省略/***<profile>*** 来安装流的这个默认配置文件。

注意

在 Red Hat Enterprise Linux 9 中，没有预定义默认的模块流。但是，如果您在模块安装过程中指定了流，则不必提前手动启用流。

例如，要从 **nodejs** 模块的流 **18** 安装默认配置文件(**common**)，请输入：

```
# dnf module install nodejs:18
=========================================================================
=========================
 Package                 Architecture   Version   Repository                 Size
=========================================================================
=========================
Installing group/module packages:
 nodejs                  x86_64         ...       rhel-9-for-x86_64-appstream-rpms    12 M
 npm                     x86_64         ...       rhel-9-for-x86_64-appstream-rpms   2.5 M
Installing weak dependencies:
 nodejs-docs             noarch         ..        rhel-9-for-x86_64-appstream-rpms   7.6 M
 nodejs-full-i18n        x86_64         ..        rhel-9-for-x86_64-appstream-rpms   8.4 M
Installing module profiles:
 nodejs/common
Enabling module streams:
 nodejs                                 18
```

验证

- 验证是否启用了正确的模块流(**[e]**)，并且安装了所需的配置文件(**[i]**)：

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

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

其它资源

- 模块
- 模块流
- 模块配置集

5.4. 定义自定义默认模块流和配置文件

Red Hat Enterprise Linux 9 不会在 AppStream 存储库中定义默认流。但是，您可以配置默认模块流和默认模块配置文件。在这种情况下，您可以在安装模块的默认流和配置文件时忽略此信息。

流程

1. 使用 **dnf module list *<module_name>*** 命令显示可用的流及其配置文件，例如：

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

在本例中，**nodejs:18** 没有被设置为默认流，此流中的默认配置文件是 **common**。

2. 在 **/etc/dnf/modules.defaults.d/** 目录中创建一个 YAML 文件，以定义模块的默认流和配置文件。
例如，使用以下内容创建 **/etc/dnf/modules.defaults.d/nodejs.yaml** 文件，来将 **18** 定义为 **nodejs** 模块的默认流，将**minimal** 定义为默认配置文件：

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

验证

- 使用 **dnf module list *<module_name>*** 命令来验证新的默认流和配置文件设置，例如：

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

其它资源

- 模块
- 模块流
- 模块配置集

5.5. 其它资源

- 在 RHEL 9 中安装内容的命令

第 6 章 更新 RHEL 9 内容

使用 DNF，您可以检查您的系统是否有任何待处理的更新。您可以列出需要更新的软件包，并选择更新单个软件包、多个软件包或者所有软件包。如果您选择要更新的软件包有依赖项，这些依赖项也会被更新。

6.1. 检查更新

要识别您系统上安装的软件包是否有可用的更新，您可以列出它们。

流程

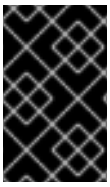
- 检查已安装的软件包的可用更新：

```
# dnf check-update
```

输出返回有可用更新的软件包及其依赖项列表。

6.2. 更新软件包

您可以使用 DNF 一次更新单个软件包、一个软件包组或所有软件包及其依赖项。



重要

当对内核应用更新时，无论是否使用了 **dnf update** 或 **dnf install** 命令，**dnf** 总会安装一个新内核。请注意，这只适用于使用 **installonlypkgs** DNF 配置选项识别的软件包。例如，这些软件包包括 **kernel**、**kernel-core** 和 **kernel-modules** 软件包。

流程

- 根据您的场景，使用以下选项之一应用更新：
  - 要更新所有软件包及其依赖项，请输入：

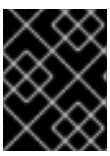
```
# dnf upgrade
```

  - 要更新单个软件包，请输入：

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

  - 要只从特定软件包组更新软件包，请输入：

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



重要

如果您在 BIOS 或 IBM Power 系统上升级了 GRUB 引导装载程序软件包，请重新安装 GRUB。请参阅 重新安装 GRUB。

6.3. 更新与安全相关的软件包

您可以使用 DNF 更新与安全相关的软件包。

流程

- 根据您的场景，使用以下选项之一应用更新：
  - 要升级到有安全勘误的最新的可用软件包，请输入：

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

  - 要升级到最后一个安全勘误软件包，请输入：

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

重要

如果您在 BIOS 或 IBM Power 系统上升级了 GRUB 引导装载程序软件包，请重新安装 GRUB。请参阅 重新安装 GRUB。

其它资源

- 管理及监控安全更新

第 7 章 在 RHEL 9 中自动化软件更新

DNF Automatic 是一个 DNF 的替代命令行界面，它适用于使用 systemd 计时器、cron 作业和其他此类工具来进行自动化和常规执行。

DNF Automatic 根据需要同步软件包元数据，检查可用的更新，然后根据您配置工具的方式执行以下操作之一：

- Exit
- 下载更新的软件包
- 下载并应用更新

然后，通过所选的机制（如标准输出或电子邮件）报告操作的结果。

7.1. 安装 DNF AUTOMATIC

要自动检查并下载软件包更新，您可以使用 **dnf-automatic** 软件包提供的 DNF Automatic 工具。

流程

- 安装 **dnf-automatic** 软件包：

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

验证

- 通过确认 **dnf-automatic** 软件包是否存在，来验证安装是否成功：

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

7.2. DNF AUTOMATIC 配置文件

默认情况下，DNF Automatic 使用 **/etc/dnf/automatic.conf** 作为其配置文件来定义其行为。

配置文件被分隔为以下主题部分：

- **[commands]**
  设置 DNF Automatic 的操作模式。

警告

[commands] 部分中的操作模式设置会被 systemd 定时器单元用于除 **dnf-automatic.timer** 之外的所有计时器单元使用的设置覆盖。

- **[emitters]**
  定义如何报告 DNF Automatic 的结果。

- **[command]**
  定义命令发出器配置。
- **[command_email]**
  为用来发送电子邮件的外部命令提供电子邮件发布程序配置。
- **[email]**
  提供电子邮件发布程序配置。
- **[base]**
  覆盖 DNF 的主要配置文件中的设置。

使用 **/etc/dnf/automatic.conf** 文件的默认设置，DNF Automatic 检查可用更新，下载它们，并将结果报告给标准输出。

其它资源

- 您系统上的 **dnf-automatic (8)** 手册页
- dnf-automatic 软件包中包含的 systemd 计时器单元概述

7.3. 启用 DNF AUTOMATIC

要运行 DNF Automatic 一次，您必须启动一个 systemd 计时器单元。但是，如果要定期运行 DNF Automatic，您必须启用计时器单元。您可以使用 **dnf-automatic** 软件包提供的计时器单元之一，或者您可以为计时器单元创建一个插入文件来调整执行时间。

先决条件

- 您可以通过修改 **/etc/dnf/automatic.conf** 配置文件来指定 DNF Automatic 的行为。

流程

- 要立即启用并执行 systemd 计时器单元，请输入：

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

  如果想只启用计时器，而不立即执行它，请省略 **--now** 选项。

  您可以使用以下计时器：

  - **dnf-automatic-download.timer**: 下载可用的更新。
  - **dnf-automatic-install.timer**: 下载并安装可用的更新。
  - **dnf-automatic-notifyonly.timer**: 报告可用的更新。
  - **dnf-automatic.timer** ：下载、下载并安装或报告可用的更新。

验证

- 验证计时器是否已启用：

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

- 可选：检查系统上每个计时器上次运行的时间：

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

其它资源

- 您系统上的 **dnf-automatic (8)** 手册页
- dnf-automatic 软件包中包含的 systemd 计时器单元概述

7.4. DNF-AUTOMATIC 软件包中包含的 SYSTEMD 计时器单元的概述

systemd 定时器单元具有优先权，并在下载和应用更新时覆盖 **/etc/dnf/automatic.conf** 配置文件中的设置。

例如，如果您在 **/etc/dnf/automatic.conf** 配置文件中设置了 **download_updates = yes**，但您已激活了 **dnf-automatic-notifyonly.timer 单元**，则不会下载软件包。

表 7.1. dnf-automatic 软件包中包含的 systemd 计时器

计时器单元	功能	覆盖 /etc/dnf/automatic.conf 文件的 [commands] 部分中的 apply_updates 和 download_updates 设置？
dnf-automatic-download.timer	下载软件包以便进行更新。 这个计时器单元没有安装更新的软件包。要执行安装，您必须运行 **dnf update** 命令。	是
dnf-automatic-install.timer	下载并安装更新的软件包。	是
dnf-automatic-notifyonly.timer	只下载存储库数据，以保持存储库缓存最新状态，并通知您是否有可用的更新。 这个计时器单元没有下载或安装更新的软件包。	是
dnf-automatic.timer	此计时器在下载和应用更新时的行为是由 **/etc/dnf/automatic.conf** 配置文件中的设置指定。 这个计时器下载软件包，但不安装它们。	否

第 8 章 删除 RHEL 9 内容

在以下部分中，了解如何使用 DNF 删除 Red Hat Enterprise Linux 9 中的内容。

8.1. 删除安装的软件包

您可以使用 DNF 删除单个软件包或安装在您的系统上的多个软件包。如果您选择要删除的软件包有未使用的依赖项，DNF 也会卸载这些依赖项。

流程

- 删除特定软件包：

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

8.2. 删除软件包组

软件包组捆绑多个软件包。您可以使用软件包组在单个步骤中删除分配给组的所有软件包。

流程

- 按组名称或组 ID 删除软件包组：

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

8.3. 删除安装的模块内容

当删除安装的模块内容时，您可以从 所选配置集 或 整个流中删除软件包。

重要

DNF 尝试删除所有名称与用安装配置文件或流安装的软件包相对应的软件包，包括其依赖软件包。在进行操作前，务必检查要删除的软件包列表，特别是您是否在系统中启用了自定义软件仓库。

8.3.1. 从安装的配置集中删除软件包

当您删除安装有配置集的软件包时，所有名称与配置集安装的软件包相对应的软件包都会被删除。这包括其依赖项，但被其他配置集所需的软件包除外。

要从所选流中删除所有软件包，请完成 从模块流中删除所有软件包 中的步骤。

先决条件

- 所选的配置文件是使用 **dnf module install *<module-name:stream/profile>*** 命令安装的或使用 **dnf install *<module-name:stream command>***作为默认配置文件安装的。

流程

- 卸载属于所选配置文件的软件包：

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

例如，要从 **nodejs:18** 模块流的 **development** 配置文件中删除软件包及其依赖项，请输入：

```
# 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
```

警告

在进行删除事务前，请检查 **Removing:** 和 **Removing unused dependencies:** 中的软件包列表。这个事务会删除请求的软件包、未使用的依赖项和依赖的软件包，这可能导致系统故障。

或者，从流中的所有安装配置集中卸载软件包：

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

注意

这些操作不会从不属于任何配置集的流中删除软件包。

验证

- 验证正确的配置文件是否已删除：

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

除了 **development**，所有配置文件目前都已安装(**[i]**)。

其它资源

- 模块依赖关系和流更改

8.3.2. 从模块流中删除所有软件包

当您删除使用模块流安装的软件包时，所有名称与流安装的软件包对应的软件包都会被删除。这包括其依赖项，但其他模块所需的软件包除外。

要从所选的配置文件中只删除软件包，请完成 从安装的配置文件中删除软件包 中的步骤。

先决条件

- 模块流已启用，并至少安装了流中的一些软件包。

流程

1. 从所选流中删除所有软件包：

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

例如，要从 **nodejs:18** 模块流中删除所有软件包，请输入：

```
# 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
```

警告

在进行删除事务前，请检查 **Removing:** 和 **Removing unused dependencies:** 中的软件包列表。这个事务会删除请求的软件包、未使用的依赖项和依赖的软件包，这可能导致系统故障。

2. 可选：输入以下命令之一来重置或禁用流：

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

验证

- 验证所选模块流中的所有软件包是否都已删除：

```
$ dnf module info nodejs
...
Name             : nodejs
```

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

其它资源

- 模块依赖关系和流更改
- 重置模块流
- 禁用一个模块的所有流

8.4. 其它资源

- 在 RHEL 9 中删除内容的命令

第 9 章 处理软件包管理历史记录

使用 **dnf history** 命令，您可以查看以下信息：

- DNF 交易的时间线。
- 交易发生的日期和时间。
- 受交易影响的软件包的数量。
- 交易是否成功或被中止。
- RPM 数据库是否在交易之间更改了。

您还可以使用 **dnf history** 命令来撤销交易。

9.1. 列出交易

您可以使用 DNF 执行以下任务：

- 列出最新的交易。
- 列出对所选软件包的最新操作。
- 显示特定交易的详细信息。

流程

- 根据您的场景，使用以下选项之一显示交易信息：
  - 要显示所有最新的 DNF 交易的列表，请输入：

    ```
    # dnf history
    ```

    输出包含以下信息：

    - **Action (s)** 列显示在交易期间执行了哪种操作类型，如 Install (**I**)、Upgrade (**U**)、删除(**E**)和其他操作。
    - **Altered** 列显示事务期间执行的操作数。操作数也可以后跟事务的结果。
      有关 **Action (s)** 和 **Altered** 列的值的更多信息，请参阅您系统上的 **dnf (8)** 手册页。
  - 要显示对所选软件包的所有最新操作的列表，请输入：

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

  - 要显示特定交易的详情，请输入：

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

注意

您可以通过将全局表达式附加为参数来过滤结果。如需了解更多详细信息，请参阅在 dnf 输入中指定全局表达式。

其它资源

- 您系统上的 **dnf (8)** 手册页

9.2. 恢复 DNF 事务

如果要撤销交易期间执行的操作，撤销 DNF 交易会很有用。例如，如果使用 **dnf install** 命令安装几个软件包，您可以通过撤销安装事务来一次卸载这些软件包。

您可以使用以下方法撤销 DNF 事务：

- 使用 **dnf history undo** 命令撤销单个 DNF 交易。
- 使用 **dnf history rollback** 命令撤销指定交易和最后一个事务之间执行的所有 DNF 交易。

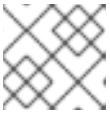
重要

不支持使用 **dnf history undo** 和 **dnf history rollback** 命令将 RHEL 系统软件包降级到旧版本。这尤其涉及 **selinux**、**selinux-policy-***、**kernel** 和 **glibc** 软件包，以及 **glibc** 的依赖项，如 **gcc**。因此，不建议将系统降级为次版本（例如，从 RHEL 9.1 降级到 RHEL 9.0），因为它可能使系统处于不正确的状态。

9.2.1. 撤销单个 DNF 交易

您可以使用 **dnf history undo** 命令撤销在单个交易中执行的步骤：

- 如果交易安装了新软件包，**dnf history undo** 会卸载软件包。
- 如果交易卸载了软件包，**dnf history undo** 会重新安装软件包。
- 如果旧的软件包可用，**dnf history undo** 命令还会尝试将所有更新的软件包降级到之前的版本。



注意

如果旧的软件包版本不可用，则使用 **dnf history undo** 命令降级会失败。

流程

1. 确定您要恢复的事务的 ID：

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

2. 可选：通过显示其详情来验证您要恢复的事务：

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

3. 恢复事务：

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

例如，如果要卸载之前安装的 **unzip** 软件包，请输入：

```
# dnf history undo 12
```

9.2.2. 撤销多个 DNF 交易

您可以使用 **dnf history rollback** 命令撤销指定交易和最后一个交易之间执行的所有 DNF 交易。请注意，交易 ID 指定的交易保持不变。

流程

1. 确定您要恢复到的状态的事务 ID：

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

2. 恢复指定的事务：

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

例如，要恢复到 **wget** 和 **unzip** 软件包安装前的状态，请输入：

```
# dnf history rollback 12
```

或者，要恢复事务历史记录中的所有事务，请使用事务 ID **1**：

```
# dnf history rollback 1
```

第 10 章 管理自定义软件存储库

您可以在 **/etc/dnf/dnf.conf** 文件或 **/etc/yum.repos.d/** 目录中的 **.repo** 文件中配置软件仓库。

重要

在 **.repo** 文件，而不是 **/etc/dnf/dnf.conf** 中定义您的存储库。

/etc/dnf/dnf.conf 文件包含 **[main]** 部分，并可包含一个或多个您可以用来设置特定于存储库的选项的存储库部分 (**[*<repository-ID>*]**)。您在 **/etc/dnf/dnf.conf** 文件的单个存储库部分定义的值会覆盖 **[main]** 部分中设置的值。

10.1. DNF 存储库选项

/etc/dnf/dnf.conf 配置文件包含存储库部分，其在括号中有一个唯一存储库 ID (**[]**)。您可以使用这些部分定义单个 DNF 存储库。

重要

[] 中的存储库 ID 必须是唯一的。

有关可用存储库 ID 选项的完整列表，请查看您系统上 **dnf.conf (5)** 手册页的 **[*<repository-ID>*] OPTIONS** 部分。

10.2. 添加 DNF 软件仓库

您可以使用 **dnf config-manager --add-repo** 命令将DNF 存储库添加到系统中。

流程

1. 在您的系统中添加软件仓库：

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

请注意，此命令添加的存储库默认被启用。

2. 检查并（可选）更新上一个命令在 **/etc/yum.repos.d/<repository_URL>.repo** 文件中创建的存储库设置：

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

警告

从基于证书的 **Content Delivery Network** (CDN)以外的未验证或不信任的源获取并安装软件包具有潜在的安全风险，并可能导致安全性、稳定性、兼容性和可维护性问题。

10.3. 启用 DNF 软件仓库

您可以使用 **dnf config-manager** 命令启用已添加到系统中的 DNF 存储库。

流程

- 启用存储库：

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

10.4. 禁用 DNF 软件仓库

您可以使用 **dnf config-manager** 命令禁用已添加到系统中的 DNF 存储库。

流程

- 禁用软件仓库：

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

第 11 章 管理应用程序流内容版本

AppStream 仓库的内容可以在多个版本中提供，对应于模块流。

11.1. 模块依赖关系和流更改

传统上，提供内容的软件包依赖于其他软件包，并且通常指定所需的依赖项版本。对于模块中包含的软件包，此机制也会应用这个机制，但将软件包及其特定版本分组到模块和流中可以进一步的限制。另外，模块流可以声明与其他模块流的依赖关系，独立于其包含的软件包并提供它们。

在使用软件包或模块操作后，所有底层已安装软件包的所有依赖项树都必须满足软件包声明的所有条件。另外，必须满足所有模块流依赖项。例如，禁用模块流可能需要禁用其他模块流。不会自动删除任何软件包。

请注意，以下操作可能导致后续的自动操作：

- 启用模块流可能导致启用其它模块流。
- 安装模块流配置文件或从流安装软件包可能导致启用其它模块流，并安装其它软件包。
- 删除软件包可能导致删除其他软件包。如果模块提供了这些软件包，则模块流仍然启用，以准备进一步安装，即使这些流中没有安装这些软件包。这会镜像未使用的 **DNF** 存储库的行为。

11.2. 模块化和非模块化依赖项的交互

模块依赖关系 是常规 RPM 依赖项之上的额外层。模块依赖关系与存储库间可能存在的依赖关系的行为相似。这意味着安装不同的软件包需要同时解析 RPM 依赖项和模块依赖关系。

系统将始终保留模块和流选择，除非明确指示要更改它们。模块软件包将接收目前启用的模块流中包含的提供此软件包的更新，但不会升级到另一流中包含的版本。

11.3. 重置模块流

重置模块是一种操作，它将此模块返回到其初始状态，既没有启用也没有禁用。如果模块有一个配置的默认流，则此流会因为重置模块而变为活动状态。

重置模块非常有用，例如，如果您希望只从模块中提取 RPM 内容而不启用该模块。在启用模块并提取其内容后，您可以使用 **dnf module reset** 命令以将此模块重置为其初始状态。

流程

- 重置模块状态：

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

  模块返回到初始状态。已启用的流和安装的配置集的信息会被清除，但没有删除安装的内容。

11.4. 禁用一个模块的所有流

具有默认流的模块始终会激活一个流。如果要使模块的所有模块流中的内容无法访问，您可以禁用整个模块。

先决条件

- 您了解了 *活动模块流*的概念。

流程

- 禁用模块：

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

dnf 命令要求确认，然后禁用该模块及其所有流。所有模块流都不再活跃。没有安装的内容被删除。

11.5. 切换到更新的流

当您切换到后期模块流时，所有对应的软件包都被后期版本替代。

重要

备份您的数据，并按照特定于组件的迁移说明进行操作。

先决条件

- 这个系统已被完全更新。

流程

1. 将安装的组件切换到新版本，并选择模块（组件）和流（版本）：

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

例如，要从 **nodejs:18** 模块流切换到 **nodejs:20** 流，请输入：

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

您还可以从非模块内容切换到模块流。例如，要从非模块 **PHP 8.0** 切换到模块 **PHP 8.1**，请输入：

```
# dnf module switch-to php:8.1
...
```

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

2. 可选：将已安装的组件切换到新版本，然后选择要安装的配置文件：

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

验证

- 验证已安装的组件是否已切换到新版本(**[e]**)：

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

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

附录 A. DNF 命令列表

在以下部分中，检查用于列出、安装和删除 Red Hat Enterprise Linux 9 上内容的 DNF 命令。

A.1. 在 RHEL 9 中列出内容的命令

以下是在 Red Hat Enterprise Linux 9 中查找内容及其详情的常用 DNF 命令：

命令	描述
dnf search *term*	使用与软件包相关的术语搜索软件包。
dnf repoquery *package*	为所选软件包及其版本搜索启用的 DNF 存储库。
dnf list	列出所有已安装的和可用的软件包信息。
dnf list --installed **dnf repoquery --installed**	列出系统上安装的所有软件包。
dnf list --available **dnf repoquery**	列出用于安装的所有已启用的存储库中的所有软件包。
dnf repolist	列出系统上所有启用的存储库。
dnf repolist --disabled	列出系统上所有禁用的存储库。
dnf repolist --all	列出启用和禁用的存储库。
dnf repoinfo	列出有关存储库的其他信息。
dnf info *package_name* **dnf repoquery --info *package_name***	显示可用软件包的详细信息。
dnf repoquery --info --installed *package_name*	显示系统上安装的软件包的详情。
dnf module list	列出模块及其当前状态。
dnf module info *module_name*	显示模块的详细信息。
dnf module list *module_name*	显示模块的当前状态。
dnf module info --profile *module_name*	显示与所选模块的可用配置文件关联的软件包。
dnf module info --profile *module_name:stream*	使用指定流显示与模块的可用配置为念关联的软件包。

命令	描述
dnf module provides *package*	确定哪些模块、流和配置集提供软件包。 请注意，如果软件包在任何模块外可用，这个命令的输出为空。
dnf group summary	查看已安装的和可用的组的数量。
dnf group list	列出所有已安装的和可用的组。
dnf group info *group_name*	列出特定组中包含的强制的和可选的软件包。

A.2. 在 RHEL 9 中安装内容的命令

以下是在 Red Hat Enterprise Linux 9 中安装内容的常用 **DNF** 命令：

命令	描述
dnf install *package_name*	安装软件包。 如果软件包由模块流提供，**dnf** 将解析所需的模块流，并在安装此软件包时自动启用它。这也会以递归方式对所有软件包依赖项进行。如果更多模块流满足要求，则使用默认模块流。
dnf install *package_name_1 package_name_2*	同时安装多个软件包及其依赖项。
dnf install *package_name.arch*	在 *multilib* 系统(AMD64、Intel 64 机器)上安装软件包时，通过将其追加到软件包名称来指定软件包的架构。
dnf install */usr/sbin/binary_file*	通过将二进制路径用作参数来安装二进制文件。
dnf install */path/*	从本地目录安装之前下载的软件包。
dnf install *package_url*	使用软件包 URL 安装远程软件包。
dnf module enable *module_name:stream*	使用特定流启用模块。 请注意，运行这个命令不会安装任何 RPM 软件包。
dnf module install *module_name:stream* **dnf install @*module_name:stream***	从特定的模块流安装默认配置集。 请注意，运行这个命令还会启用指定的流。

命令	描述
dnf module install *module_name:stream/profile* **dnf install @*module_name:stream/profile***	使用特定流安装所选配置文件。
dnf group install *group_name*	按组名称安装软件包组。
dnf group install *group_ID*	按 groupID 安装软件包组。

A.3. 在 RHEL 9 中删除内容的命令

以下是删除 Red Hat Enterprise Linux 9 中内容的常用 **DNF** 命令：

命令	描述
dnf remove *package_name*	删除特定软件包以及所有依赖的软件包。
dnf remove *package_name_1 package_name_2*	同时删除多个软件包及其未使用的依赖项。
dnf group remove group_name	按组名称删除软件包组。
dnf group remove *group_ID*	按 groupID 删除软件包组。
dnf module remove --all *module_name:stream*	从指定的流中删除所有软件包。 请注意，运行此命令可从系统中删除关键软件包。
dnf module remove *module_name:stream/profile*	从安装的配置文件删除软件包。
dnf module remove *module_name:stream*	从指定流中的所有安装的配置文件删除软件包。
dnf module reset *module_name*	将模块重置为初始状态。 请注意，运行此命令不会从指定的模块中删除软件包。
dnf module disable *module_name*	禁用一个模块及其所有流。 请注意，运行此命令不会从指定的模块中删除软件包。