Red Hat Enterprise Linux 9

DNF ツールを使用したソフトウェアの管理

DNF ソフトウェア管理ツールを使用した RPM リポジトリー内のコンテンツの管理

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 DNF ツールを使用したソフトウェアの管理

DNF ソフトウェア管理ツールを使用した RPM リポジトリー内のコンテンツの管理

法律上の通知

概要

DNF ツールを使用して、RPM リポジトリーを通じて配布されたコンテンツを検索、インストール、利用します。パッケージ、モジュール、ストリーム、プロファイルの操作方法を学びます。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

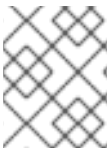
Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 RED HAT ENTERPRISE LINUX 9 のソフトウェア管理ツール

Red Hat Enterprise Linux (RHEL) 9 では、**DNF** ユーティリティーを使用してソフトウェアを管理します。以前のメジャー RHEL バージョンとの互換性を確保するために、**yum** コマンドも引き続き使用できます。ただし、RHEL 9 の **yum** は、**yum** とある程度の互換性を持つ **dnf** のエイリアスです。



注記

RHEL 8 と RHEL 9 は**DNF**をベースにしていますが、RHEL 7 で使用していた**YUM**との互換性があります。

第2章 RHEL 9 のコンテンツの配布

以下のセクションでは、Red Hat Enterprise Linux 9 でのソフトウェアの配布方法を説明します。

2.1. リポジトリー

Red Hat Enterprise Linux (RHEL) は、次のようなさまざまなリポジトリーを通じてコンテンツを配信します。

BaseOS

BaseOS リポジトリーのコンテンツは、すべてのインストールの基盤を提供する、基礎となるオペレーティングシステム機能のコアセットで構成されています。このコンテンツは RPM 形式で提供されており、RHEL の以前のリリースと同様のサポート条件が適用されます。

AppStream

AppStream リポジトリーには、さまざまなワークロードとユースケースに対応するために、ユーザー空間アプリケーション、ランタイム言語、およびデータベースが同梱されます。

重要

BaseOS コンテンツセットと AppStream コンテンツセットは両方とも RHEL に必要であり、すべての RHEL サブスクリプションで利用できます。

CodeReady Linux Builder

CodeReady Linux Builder リポジトリーは、すべての RHEL サブスクリプションで利用できます。このリポジトリーは、開発者向けの追加パッケージを提供します。Red Hat は、CodeReady Linux Builder リポジトリーに含まれるパッケージをサポートしていません。

関連情報

- パッケージマニフェスト

2.2. APPLICATION STREAMS (APPSTREAM)

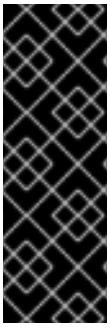
Red Hat は、複数のバージョンのユーザー空間コンポーネントを Application Streams として提供しています。Application Streams は、コアオペレーティングシステムパッケージよりも頻繁に更新されます。これにより、プラットフォームや特定デプロイメントの基本的な安定性に影響を与えることなく、Red Hat Enterprise Linux をより柔軟にカスタマイズできます。

Application Streams は以下の形式で利用できます。

- RPM 形式
- RPM 形式の拡張であるモジュール
- Software Collections

RHEL 9 では、Application Streams の初期バージョンを RPM として提供することで、Application Streams のエクスペリエンスを向上させています。RPM は、**dnf install** コマンドを使用してインストールできます。

RHEL 9.1 以降、Red Hat は、より短いライフサイクルのモジュールとして追加の Application Streams バージョンを提供しています。



重要

各 Application Streams には、RHEL 9 と同じか、それより短い独自のライフサイクルがあります。Red Hat Enterprise Linux Application Streams Life Cycle を参照してください。

インストールする必要がある Application Stream のバージョンを常に確認してください。その際には、まず RHEL Application Stream のライフサイクルを必ず確認してください。

関連情報

- Red Hat Enterprise Linux 9:アプリケーションの互換性ガイド
- パッケージマニフェスト
- Red Hat Enterprise Linux Application Streams Life Cycle

2.3. モジュール

モジュールは、コンポーネントを表す RPM パッケージのセットです。一般的なモジュールには次のパッケージタイプが含まれます。

- アプリケーションを含むパッケージ
- アプリケーション固有の依存関係ライブラリーを含むパッケージ
- アプリケーションのドキュメントを含むパッケージ
- ヘルパーユーティリティーを含むパッケージ

2.4. モジュールストリーム

モジュールのストリームは、AppStream 物理リポジトリーで仮想的なリポジトリーとして扱えるフィルターです。モジュールストリームは、AppStream コンポーネントのバージョンです。各ストリームは別々に更新を受信し、他のモジュールストリームに依存している場合があります。

モジュールストリームは、アクティブまたは非アクティブにできます。アクティブなストリームにより、システムは特定のモジュールストリーム内の RPM パッケージにアクセスできるようになり、それぞれのコンポーネントバージョンをインストールできるようになります。

ストリームは次の場合にアクティブになります。

- 管理者が明示的に有効にした場合。
- ストリームが有効なモジュールの依存関係である場合。
- ストリームがデフォルトのストリームの場合。各モジュールにはデフォルトのストリームを設定できますが、Red Hat Enterprise Linux 9 ではデフォルトのストリームは定義されていません。必要に応じて、カスタムのデフォルトモジュールストリームとプロファイルの定義 の説明に従ってデフォルトストリームを設定できます。

あるモジュールの中で同時にアクティブにできるストリームは 1 つだけです。したがって、利用できるのは特定のストリームのパッケージだけです。

ランタイムユーザーアプリケーションまたは開発者アプリケーションに特定のストリームを選択する前に、次の点を考慮してください。

- 必要な機能と、その機能に対応するコンポーネントのバージョン
- アプリケーションとの互換性やユースケース
- Application Streams の ライフサイクル と更新計画

利用可能なモジュールおよびストリームのリストは パッケージマニフェスト を参照してください。コンポーネントごとの変更は、リリースノート を参照してください。

関連情報

- モジュールの依存関係とストリームの変更

2.5. モジュールプロファイル

モジュールプロファイルは、サーバー、クライアント、開発、最小インストールなど、特定のユースケースでまとめてインストールされる推奨パッケージのリストです。このパッケージリストは、モジュールストリームに含まれないパッケージ (通常は BaseOS リポジトリー、またはそのストリームの依存関係) も含みます。

利便性のため、プロファイル (一度の操作で完了) を利用してパッケージのインストールを行えます。同じモジュールストリームから複数のプロファイルを利用してパッケージをインストールすることも、特に準備などを行わずに行えます。

各モジュールストリームではプロファイル数に制限がありません (ゼロにすることも可能)。特定のモジュールストリームについて、そのプロファイルの一部を **デフォルト** としてマークできます。デフォルトプロファイルは、プロファイルを明示的に指定しなかった場合、プロファイルのインストールアクションに使用されます。ただし、モジュールストリームのデフォルトプロファイルは必須ではありません。

> **例2.1 nodejs モジュールプロファイル**
>
> Node.js ランタイム環境を提供する **nodejs** モジュールは、インストール用に次のプロファイルを提供します。
>
> ```
> # 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
> ```
>
> この例では、次のプロファイルが利用可能です。
>
> - **common**:実稼働環境に対応したパッケージ。これはデフォルトのプロファイル (**d**) です。
> - **development**:Node.js 開発ヘッダーを含む、実稼働環境に対応したパッケージ。
> - **minimal**:Node.js ランタイム環境を提供するパッケージの最小セット。
> - **s2i**:Node.js Source-to-Image (S2I) Linux コンテナーの作成に必要なパッケージ。

第3章 DNF の設定

DNF および関連ユーティリティーの設定は、**/etc/dnf/dnf.conf** ファイルの **[main]** セクションに保存されます。

3.1. 現在の DNF 設定の表示

/etc/dnf/dnf.conf ファイルの **[main]** セクションには、明示的に指定した設定のみが含まれます。ただし、**[main]** セクションの設定はすべて表示することができます。これには、未設定でデフォルト値が使用されている設定も含まれます。

手順

- グローバルの DNF 設定を表示します。

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

関連情報

- **dnf.conf(5)** man ページ

3.2. DNF メインオプションの設定

/etc/dnf/dnf.conf ファイルには、1 つの **[main]** セクションが含まれています。このセクションのキーと値のペアは、DNF のリポジトリーの操作方法および処理方法に影響を及ぼします。

手順

1. **/etc/dnf/dnf.conf** ファイルを編集します。
2. 要件に応じて **[main]** セクションを更新します。
3. 変更を保存します。

関連情報

- **dnf.conf(5)** man ページの **[main] OPTIONS** および **OPTIONS FOR BOTH [main] AND REPO** セクション

3.3. DNF プラグインの管理

インストールした各プラグインの **/etc/dnf/plugins/** ディレクトリーには、それぞれ独自の設定ファイルが含まれている場合があります。このディレクトリー内のプラグイン設定ファイルには、**<plug-in_name>.conf** という名前を付けます。通常、デフォルトではプラグインが有効になっています。これらの設定ファイルのいずれかでプラグインを無効にするには、ファイルに次の内容を追加します。

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

3.4. DNF プラグインの有効化および無効化

DNF ツールでは、プラグインがデフォルトでロードされます。ただし、DNF がロードするプラグインは制御することができます。

警告

すべてのプラグインの無効化は、潜在的な問題を診断する場合にのみ行ってください。DNF には **product-id** や **subscription-manager** などの特定のプラグインが必要です。それらを無効にすると、Red Hat Enterprise Linux はコンテンツ配信ネットワーク (CDN) からソフトウェアをインストールまたは更新できなくなります。

手順

- 次のいずれかの方法を使用して、DNF がプラグインを使用する方法を制御します。
  - DNF プラグインのロードをシステム全体で有効または無効にするには、**/etc/dnf/dnf.conf** ファイルの **[main]** セクションに **plugins** パラメーターを追加します。
    - すべての DNF プラグインのロードを有効にするには、**plugins=1** (デフォルト) を設定します。
    - すべての DNF プラグインのロードを無効にするには、**plugins=0** を設定します。
  - 特定のプラグインを無効にするには、**/etc/dnf/plugins/<plug-in_name>.conf** ファイルの **[main]** セクションに **enabled=False** を追加します。
  - 特定のコマンドですべての DNF プラグインを無効にするには、コマンドに **--noplugins** オプションを追加します。たとえば、1 つの update コマンドで DNF プラグインを無効にするには、次のように入力します。

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

  - 1 つのコマンドで特定の DNF プラグインを無効にするには、コマンドに **--disableplugin=plugin-name** オプションを追加します。たとえば、1 つの update コマンドで特定の DNF プラグインを無効にするには、次のように入力します。

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

  - 1 つのコマンドで特定の DNF プラグインを有効にするには、コマンドに **--enableplugin=plugin-name** オプションを追加します。たとえば、1 つの update コマンドで特定の DNF プラグインを有効にするには、次のように入力します。

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

第4章 RHEL 9 コンテンツの検索

以下のセクションでは、**DNF** を使用して、Red Hat Enterprise Linux 9 の AppStream および BaseOS リポジトリー内のコンテンツを見つけて調べる方法を説明します。

4.1. ソフトウェアパッケージの検索

必要なソフトウェアを提供するパッケージを特定するには、**DNF** を使用してリポジトリーを検索します。

手順

- シナリオに応じて、次のいずれかのオプションを使用してリポジトリーを検索します。
  - パッケージの名前または概要内の用語を検索するには、次のように入力します。

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

  - パッケージの名前、概要、または説明内の用語を検索するには、次のように入力します。

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

    --all オプションを使用して説明内で追加の検索を行うと、通常の検索操作よりも時間がかかることに注意してください。

  - パッケージ名を検索し、出力にパッケージ名とそのバージョンをリストするには、次のように入力します。

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

  - ファイルを提供するパッケージを検索するには、ファイル名またはファイルへのパスを指定します。

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

4.2. ソフトウェアパッケージのリスト表示

DNF を使用すると、リポジトリーで使用可能なパッケージとそのバージョンのリストを表示できます。必要に応じてこのリストをフィルタリングして、たとえば更新を利用できるパッケージのみをリストすることができます。

手順

- 利用可能なすべてのパッケージの最新バージョンを、アーキテクチャー、バージョン番号、インストール元のリポジトリーを含めてリスト表示します。

  ```
  $ 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
  ...
  ```

リポジトリーの前の @ 記号は、この行のパッケージが現在インストールされていることを示します。

または、使用可能なすべてのパッケージを、バージョン番号とアーキテクチャーを含めて表示するには、次のように入力します。

```
$ 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
...
```

必要に応じて、**--all** の代わりに他のオプションを使用して出力をフィルタリングできます。次に例を示します。

- インストールされているパッケージのみをリスト表示するには、**--installed** を使用します。
- 利用可能なすべてのパッケージをリスト表示するには、**--available** を使用します。
- 新しいバージョンを利用できるパッケージをリスト表示するには、**--upgrades** を使用します。

注記

glob 表現を引数として追加することで、結果をフィルタリングできます。詳細は、DNF 入力での glob 表現の指定 を参照してください。

4.3. リポジトリーのリスト表示

システムで有効または無効になっているリポジトリーの概要を取得するには、リポジトリーをリスト表示します。

手順

1. システムで有効になっているすべてのリポジトリーをリスト表示します。

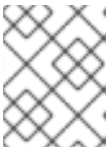
```
$ dnf repolist
```

特定のリポジトリーのみを表示するには、次のいずれかのオプションをコマンドに追加します。

- **--disabled** を追加すると、無効なリポジトリーのみがリストされます。
- **--all** を追加すると、有効なリポジトリーと無効なリポジトリーの両方がリストされます。

2. オプション: リポジトリーの追加情報をリストします。

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



注記

グローバル式を使用して結果をフィルタリングできます。詳細は、DNF 入力での glob 表現の指定 を参照してください。

4.4. パッケージ情報の表示

DNF リポジトリーをクエリーして、次のようなパッケージに関する詳細を表示できます。

- バージョン
- リリース
- アーキテクチャー
- パッケージサイズ
- 説明

手順

- 1 つ以上の利用可能なパッケージに関する情報を表示します。

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

このコマンドは、現在インストールされているパッケージの情報と、リポジトリー内にあるその新しいバージョン (利用可能な場合) の情報を表示します。または、次のコマンドを使用して、指定した名前を持つ、リポジトリー内のすべてのパッケージの情報を表示します。

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

注記

glob 表現を引数として追加することで、結果をフィルタリングできます。詳細は、DNF 入力での glob 表現の指定 を参照してください。

4.5. パッケージグループとグループが提供するパッケージのリスト表示

パッケージグループには複数のパッケージをバンドルされています。パッケージグループを使用すると、グループに割り当てられたすべてのパッケージを 1 回の手順でインストールできます。ただし、インストールする前に、必要なパッケージグループの名前を特定する必要があります。

手順

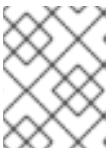
1. インストールされているグループと使用可能なグループの両方をリストします。

```
$ dnf group list
```

dnf group list コマンドに **--installed** および **--available** オプションを追加すると、結果をフィルタリングできます。**--hidden** オプションを使用すると、出力に非表示のグループを表示できます。

2. 特定のグループに含まれる必須、オプション、およびデフォルトのパッケージをリストします。

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



注記

glob 表現を引数として追加することで、結果をフィルタリングできます。詳細は、DNF 入力での glob 表現の指定 を参照してください。

3. オプション: インストールされているグループと利用可能なグループの数を表示します。

```
$ dnf group summary
```

4.6. 利用可能なモジュールとその内容の一覧表示

DNF を使用してモジュールを検索し、モジュールに関する情報を表示すると、モジュールをインストールする前に、リポジトリーで使用可能なモジュールを特定し、適切なストリームを選択できます。

手順

1. 次のいずれかの方法でモジュール情報をリスト表示します。

- 利用可能なすべてのモジュールをリストします。

```
$ 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
```

特定のモジュールのみを対象に同じ情報をリスト表示するには、**dnf module list <module_name>** コマンドを使用します。

- どのモジュールが特定のパッケージを提供しているかを検索します。

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

たとえば、**npm** パッケージを提供するモジュールとプロファイルを表示するには、次のように入力します。

```
# 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. モジュールの詳細をリスト表示するには、次のいずれかの方法を使用します。

- 説明、すべてのプロファイルのリスト、モジュールが提供するすべてのパッケージのリス

- 説明、すべてのプロファイルのリスト、モジュールが提供するすべてのパッケージのリストなど、モジュールに関するすべての詳細をリストします。

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

たとえば、**nodejs** パッケージに関する詳細を表示するには、次のように入力します。

```
$ 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
...
```

- 各モジュールプロファイルがインストールするパッケージをリストします。

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

たとえば、**nodejs** モジュールのこの情報を表示するには、次のように入力します。

```
$ 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
...
```

関連情報

- モジュール
- モジュールストリーム
- モジュールプロファイル

4.7. DNF 入力での GLOB 表現の指定

1 つ以上の glob 表現を引数として追加することで、**dnf** コマンドの結果をフィルタリングできます。

手順

- **dnf** コマンドで glob 表現を使用する場合は、次のいずれかの方法を使用します。
  - glob 表現全体を一重引用符または二重引用符で囲みます。

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

    <file_name> の前に、絶対パスの場合は / を付ける必要があります。フルパスが不明な場合にワイルドカードを使用するには、*/ を付ける必要があります。

  - ワイルドカード文字の前にはバックスラッシュ (\) を追加して、ワイルドカード文字をエスケープします。

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

4.8. 関連情報

- RHEL 9 でコンテンツをリスト表示するコマンド

第5章 RHEL 9 コンテンツのインストール

以下のセクションでは、**DNF** を使用して Red Hat Enterprise Linux 9 のコンテンツをインストールする方法を説明します。

5.1. パッケージのインストール

ソフトウェアがデフォルトのインストールに含まれていない場合は、手動でインストールできます。**DNF** は依存関係を自動的に解決してインストールします。

前提条件

- オプション: インストールするパッケージの名前がわかっている。
- インストールするパッケージがモジュールストリームによって提供されている場合、それぞれのモジュールストリームが有効になっている。

手順

- 次のいずれかの方法を使用してパッケージをインストールします。
  - リポジトリーからパッケージをインストールするには、次のように入力します。

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

    i686 や **x86_64** などの複数のアーキテクチャーをサポートするシステムにパッケージをインストールする場合は、パッケージ名にそれを追加することで、パッケージのアーキテクチャーを指定できます。

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

  - パッケージが提供するファイルへのパスだけがわかっていて、パッケージ名がわからない場合にパッケージをインストールするには、次のパスを使用して対応するパッケージをインストールできます。

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

  - ローカル RPM ファイルをインストールするには、次のように入力します。

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

    パッケージに依存関係がある場合は、これらの RPM ファイルへのパスも指定します。指定しなかった場合、**DNF** はリポジトリーから依存関係をダウンロードします。依存関係がリポジトリーで利用できない場合は失敗します。

関連情報

- モジュールコンテンツのインストール

5.2. パッケージグループのインストール

パッケージグループには複数のパッケージをバンドルされています。パッケージグループを使用すると、グループに割り当てられたすべてのパッケージを 1 回の手順でインストールできます。

前提条件

- インストールするグループの名前または ID がわかっている。

手順

- パッケージグループをインストールします。

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

5.3. モジュールコンテンツのインストール

特定のソフトウェアについては、Red Hat がモジュールを提供しています。モジュールを使用して、特定のバージョン (ストリーム) とパッケージのセット (プロファイル) をインストールできます。

手順

1. インストールするパッケージを提供するモジュールをリスト表示します。

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

たとえば、**nodejs** モジュールの詳細をリスト表示するには、次のように入力します。

```
# 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. モジュールをインストールします。

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

ストリームのデフォルトプロファイルが定義されている場合は、コマンドの **/<profile>** を省略して、このストリームのデフォルトプロファイルをインストールできます。

注記

Red Hat Enterprise Linux 9 では、デフォルトのモジュールストリームは事前定義されていません。ただし、前述のようにモジュールのインストール時にストリームを指定する場合は、事前にストリームを手動で有効にする必要はありません。

たとえば、**nodejs** モジュールのストリーム **18** からデフォルトのプロファイル (**common**) をインストールするには、次のように入力します。

```
# 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
```

検証

- 正しいモジュールストリームが有効になっており (**[e]**)、必要なプロファイルがインストールされている (**[i]**) ことを確認します。

```
# 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
```

関連情報

- モジュール
- モジュールストリーム
- モジュールプロファイル

5.4. カスタムのデフォルトモジュールストリームとプロファイルの定義

Red Hat Enterprise Linux 9 では、AppStream リポジトリーにデフォルトのストリームが定義されていません。ただし、デフォルトのモジュールストリームとデフォルトのモジュールプロファイルを設定できます。設定すると、デフォルトのモジュールのストリームとプロファイルをインストールするときに、その情報を省略できます。

手順

1. **dnf module list <module_name>** コマンドを使用して、利用可能なストリームとそのプロファイルを表示します。次に例を示します。

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

この例では、**nodejs:18** はデフォルトストリームとして設定されておらず、このストリームのデフォルトプロファイルは **common** です。

2. **/etc/dnf/modules.defaults.d/** ディレクトリーに YAML ファイルを作成して、モジュールのデフォルトのストリームとプロファイルを定義します。
   たとえば、次の内容を含む **/etc/dnf/modules.defaults.d/nodejs.yaml** ファイルを作成して、**nodejs** モジュールのデフォルトストリームとして **18** を定義し、デフォルトプロファイルとして **minimum** を定義します。

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

検証

- **dnf module list <module_name>** コマンドを使用して、新しいデフォルトのストリームとプロファイルの設定を確認します。次に例を示します。

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

関連情報

- モジュール
- モジュールストリーム
- モジュールプロファイル

5.5. 関連情報

- RHEL 9 にコンテンツをインストールするコマンド

第6章 RHEL 9 コンテンツの更新

DNF では、システムに保留中の更新があるかどうかを確認できます。更新が必要なパッケージをリスト表示して、1つのパッケージ、複数のパッケージ、またはすべてのパッケージを一度に更新できます。更新を選択したパッケージに依存関係がある場合は、これらの依存関係も更新されます。

6.1. 更新の確認

システムにインストールされているパッケージに利用可能な更新があるかどうかを識別するには、それらをリスト表示します。

手順

- インストールされたパッケージの利用可能な更新を確認します。

```
# dnf check-update
```

このコマンドは、更新が利用可能なパッケージおよびその依存関係のリストを表示します。

6.2. パッケージの更新

DNF を使用すると、単一のパッケージ、パッケージグループ、またはすべてのパッケージとその依存関係を一度に更新できます。

重要

カーネルの更新を適用する際に、**dnf** は、**dnf upgrade** コマンドまたは **dnf install** コマンドを使用しているかどうかに関わらず、新しいカーネルを常にインストールします。これは、**installonlypkgs** DNF 設定オプションを使用して識別されたパッケージにのみ適用されることに注意してください。このようなパッケージには、たとえば、**kernel**、**kernel-core**、および **kernel-modules** パッケージが含まれます。

手順

- シナリオに応じて、次のいずれかのオプションを使用して更新を適用します。
  - すべてのパッケージとその依存関係を更新するには、次のコマンドを実行します。

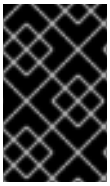
```
# dnf upgrade
```

  - 単一のパッケージを更新するには、次のように入力します。

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

  - 特定のパッケージグループからのパッケージのみを更新するには、次のように実行します。

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



重要

BIOS または IBM Power システムで GRUB ブートローダーパッケージをアップグレードした場合は、GRUB を再インストールします。GRUB の再インストール を参照してください。

6.3. セキュリティー関連パッケージの更新

DNF を使用して、セキュリティー関連のパッケージを更新できます。

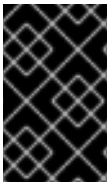
手順

- シナリオに応じて、次のいずれかのオプションを使用して更新を適用します。
  - セキュリティーエラータが含まれる、利用可能な最新パッケージにアップグレードするには以下を実行します。

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

  - 最新のセキュリティーエラータパッケージにアップグレードするには、以下を実行します。

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



重要

BIOS または IBM Power システムで GRUB ブートローダーパッケージをアップグレードした場合は、GRUB を再インストールします。GRUB の再インストール を参照してください。

関連情報

- セキュリティー更新の管理および監視

第7章 RHEL 9 でのソフトウェア更新の自動化

DNF Automatic は、DNF に対する代替のコマンドラインインターフェイスで、systemd タイマーや cron ジョブなどのツールを使用した自動実行や定期実行に適しています。

DNF Automatic は、必要に応じてパッケージメタデータを同期し、利用可能な更新を確認してから、ツールの設定方法に応じて以下のアクションのいずれかを実行します。

- 終了
- 更新済みパッケージのダウンロード
- 更新のダウンロードおよび適用

その後、標準出力やメールなど、選択したメカニズムによって操作の結果が報告されます。

7.1. DNF AUTOMATIC のインストール

パッケージの更新を自動的かつ定期的に確認してダウンロードするには、**dnf-automatic** パッケージに含まれる DNF Automatic ツールを使用できます。

手順

- **dnf-automatic** パッケージをインストールします。

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

検証

- **dnf-automatic** パッケージが存在することを確認して、インストールが正常に完了したことを確認します。

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

7.2. DNF AUTOMATIC 設定ファイル

初期設定では、DNF Automatic は **/etc/dnf/automatic.conf** を設定ファイルとして使用し、動作を定義します。

設定ファイルは、以下のトピックセクションに分かれています。

- **[commands]**
  DNF Automatic の操作モードを設定します。

警告

[commands] セクションの操作モードの設定は、**dnf-automatic.timer** 以外のすべてのタイマーユニットに対して、systemd タイマーユニットで使用される設定によって上書きされます。

- **[emitters]**
  DNF Automatic の結果が報告される方法を定義します。
- **[command]**
  コマンドエミッター設定を定義します。
- **[command_email]**
  電子メールの送信に使用する外部コマンドのメールエミッター設定を提供します。
- **[email]**
  電子メールエミッターの設定を提供します。
- **[base]**
  DNF のメイン設定ファイルの設定を上書きします。

/etc/dnf/automatic.conf ファイルのデフォルト設定では、**DNF Automatic** は利用可能な更新を確認し、ダウンロードして、標準出力に結果を報告します。

関連情報

- システムの **dnf-automatic(8)** man ページ
- dnf-automatic パッケージに含まれる systemd タイマーユニットの概要

7.3. DNF AUTOMATIC の有効化

DNF Automatic を 1 回実行するには、systemd タイマーユニットを起動する必要があります。ただし、**DNF Automatic** を定期的に実行する場合は、タイマーユニットを有効にする必要があります。**dnf-automatic** パッケージで提供されているタイマーユニットの 1 つを使用することも、タイマーユニットのドロップインファイルを作成して実行時間を調整することもできます。

前提条件

- **/etc/dnf/automatic.conf** 設定ファイルを変更して、**DNF Automatic** の挙動を指定している。

手順

- systemd タイマーユニットをすぐに有効にして実行するには、次のように入力します。

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

  タイマーをすぐに実行せずに有効にするだけの場合は、**--now** オプションを省略します。

  次のタイマーを使用できます。

  - **dnf-automatic-download.timer**:利用可能な更新をダウンロードします。
  - **dnf-automatic-install.timer**:利用可能な更新をダウンロードしてインストールします。
  - **dnf-automatic-notifyonly.timer**:利用可能な更新を報告します。
  - **dnf-automatic.timer**:利用可能な更新プログラムをダウンロード、ダウンロードしてインストール、または報告します。

検証

- タイマーが有効化されていることを確認します。

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

- オプション: システム上の各タイマーが最後に実行された時刻を確認します。

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

関連情報

- **dnf-automatic (8)** man ページ
- dnf-automatic パッケージに含まれる systemd タイマーユニットの概要

7.4. DNF-AUTOMATIC パッケージに含まれる SYSTEMD タイマーユニットの概要

更新のダウンロードおよび適用時に、systemd タイマーユニットが優先され、**/etc/dnf/automatic.conf** 設定ファイルの設定が上書きされます。

たとえば、**/etc/dnf/automatic.conf** 設定ファイルで **download_updates = yes** と設定していても、**dnf-automatic-notifyonly.timer** ユニットをアクティブにした場合は、パッケージはダウンロードされません。

表7.1 dnf-automatic に含まれる systemd タイマー

タイマーユニット	機能	/etc/dnf/automatic.conf ファイルの [commands] セクションの apply_updates および download_updates 設定を上書きしますか?
dnf-automatic-download.timer	キャッシュにパッケージをダウンロードし、更新を利用できるようにします。 このタイマーユニットでは更新パッケージはインストールされません。インストールを実行するには、**dnf update** コマンドを実行する必要があります。	はい
dnf-automatic-install.timer	更新したパッケージをダウンロードしてインストールします。	はい
dnf-automatic-notifyonly.timer	リポジトリーデータのみをダウンロードして、リポジトリーキャッシュを最新の状態に保ち、利用可能な更新を通知します。 このタイマーユニットでは、更新されたパッケージはダウンロードまたはインストールされません。	はい

タイマーユニット	機能	/etc/dnf/automatic.conf ファイルの [commands] セクションの apply_updates および download_updates 設定を上書きしますか?
dnf-automatic.timer	更新のダウンロードおよび適用時のこのタイマーの動作は、**/etc/dnf/automatic.conf** 設定ファイルの設定により指定されます。 このタイマーはパッケージをダウンロードしますが、インストールはしません。	No

第8章 RHEL 9 コンテンツの削除

以下のセクションでは、DNF を使用して Red Hat Enterprise Linux 9 のコンテンツを削除する方法を説明します。

8.1. インストール済みパッケージの削除

DNF を使用すると、システムにインストールされている単一のパッケージまたは複数のパッケージを削除できます。削除を選択したパッケージに未使用の依存関係がある場合、DNF はこれらの依存関係もアンインストールします。

手順

- 特定のパッケージを削除します。

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

8.2. パッケージグループの削除

パッケージグループは複数のパッケージをバンドルします。パッケージグループを使用すると、グループに割り当てられているすべてのパッケージを 1 つの手順で削除できます。

手順

- グループ名またはグループ ID でパッケージグループを削除します。

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

8.3. インストールしたモジュラーコンテンツの削除

インストールしたモジュールコンテンツを削除する場合は、選択したプロファイル または 全ストリーム からパッケージを削除できます。

重要

DNF は、プロファイルまたはストリームでインストールされたパッケージに対応する名前を持つパッケージ (依存パッケージを含む) をすべて削除しようとします。(特にシステムでカスタムリポジトリーを有効にしている場合は) 続行する前に、削除するパッケージのリストを常に確認してください。

8.3.1. インストール済みプロファイルからのパッケージの削除

プロファイルとともにインストールしたパッケージを削除すると、そのプロファイルによりインストールされたパッケージに対応する名前を持つパッケージがすべて削除されます。これには、依存関係も含まれます。ただし、別のプロファイルで必要とされるパッケージは除外されます。

選択したストリームからすべてのパッケージを削除するには、モジュールストリームからすべてのパッケージを削除する の手順を完了します。

前提条件

- 選択したプロファイルが、**dnf module install <module-name:stream/profile>** コマンドを使用してインストールされているか、**dnf install <module-name:stream command>** を使用してデフォルトプロファイルとしてインストールされています。

手順

- 選択したプロファイルに属するパッケージをアンインストールします。

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

たとえば、**nodejs:18** モジュールストリームの **development** プロファイルからパッケージとその依存関係を削除するには、次のように実行します。

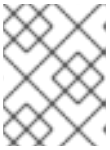
```
# dnf module remove nodejs:18/development
(...)
Dependencies resolved.
================================================================================

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

削除トランザクションに進む前に、**Removing:** および **Removing unused dependencies:** にあるパッケージのリストを確認してください。このトランザクションにより、要求されたパッケージ、未使用の依存関係、および依存パッケージが削除され、システム障害が発生する可能性があります。

または、ストリームにインストールされているすべてのプロファイルからパッケージをアンインストールします。

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



注記

この操作は、プロファイルに属さないパッケージをストリームから削除しません。

検証

- 正しいプロファイルが削除されたことを確認します。

```
$ 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
```

development を除くすべてのプロファイルが現在インストール (**[i]**) されています。

関連情報

- モジュールの依存関係とストリームの変更

8.3.2. モジュールストリームから全パッケージを削除

モジュールストリームでインストールしたパッケージを削除すると、そのストリームによりインストールされたパッケージに対応する名前を持つパッケージがすべて削除されます。これには、その依存関係も含まれます。ただし、その他のモジュールで必要なパッケージは除外されます。

選択したプロファイルからパッケージのみを削除するには、インストールされたプロファイルからパッケージを削除する の手順を完了します。

前提条件

- モジュールストリームが有効になり、少なくともいくつかのパッケージがストリームからインストールされている。

手順

1. 選択したストリームからパッケージをすべて削除します。

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

たとえば、**nodejs:18** モジュールストリームからすべてのパッケージを削除するには、次のように入力します。

```
# 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

削除トランザクションに進む前に、**Removing:** および **Removing unused dependencies:** にあるパッケージのリストを確認してください。このトランザクションにより、要求されたパッケージ、未使用の依存関係、および依存パッケージが削除され、システム障害が発生する可能性があります。

2. オプション: 次のいずれかのコマンドを実行して、ストリームをリセットまたは無効にします。

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

検証

- 選択したモジュールストリームからすべてのパッケージが削除されたことを確認します。

```
$ 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
```

関連情報

- モジュールの依存関係とストリームの変更
- モジュールストリームのリセット
- モジュールのストリームをすべて無効化

8.4. 関連情報

- RHEL 9 でコンテンツを削除するコマンド

第9章 パッケージ管理履歴の処理

dnf history コマンドを使用すると、以下の情報を確認できます。

- DNF トランザクションのタイムライン
- トランザクションの発生日時
- トランザクションの影響を受けたパッケージの数
- トランザクションの成功または中止の有無
- トランザクション間で RPM データベースが変更された場合

dnf history コマンドを使用して、トランザクションをやり直すことができます。

9.1. トランザクションのリスト表示

DNF を使用して次のタスクを実行できます。

- 最新のトランザクションをリスト表示します。
- 選択したパッケージの最新の操作をリスト表示します。
- 特定のトランザクションの詳細を表示します。

手順

- シナリオに応じて、次のいずれかのオプションを使用してトランザクション情報を表示します。
  - 最新の DNF トランザクションのリストを表示するには、以下のコマンドを実行します。

    ```
    # dnf history
    ```

    出力には、以下の情報が含まれます。

    - **Action(s)** 列には、トランザクション中に実行されたアクションのタイプが表示されます (例: インストール (**I**)、アップグレード (**U**)、削除 (**E**) など)。
    - **Altered** 列には、トランザクション中に実行されたアクションの数が表示されます。アクションの数の後に、トランザクションの結果を指定することもできます。
      Action(s) および **Altered** 列の値の詳細は、**dnf(8)** の man ページを参照してください。
  - 選択したパッケージの最新操作のリストを表示するには、以下を実行します。

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

  - 特定のトランザクションの詳細を表示するには、以下のコマンドを実行します。

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

注記

glob 表現を引数として追加することで、結果をフィルタリングできます。詳細は、dnf input での glob 表現の指定 を参照してください。

関連情報

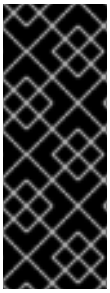
- **dnf(8)** の man ページ

9.2. DNF トランザクションの取り消し

DNF トランザクションを元に戻すことは、トランザクション中に実行された操作を元に戻す場合に役立ちます。たとえば、**dnf install** コマンドを使用して複数のパッケージをインストールした場合は、インストールトランザクションを元に戻すことで、これらのパッケージを一度にアンインストールできます。

DNF トランザクションは次の方法で元に戻すことができます。

- **dnf history undo** コマンドを使用して、単一の DNF トランザクションを元に戻します。
- **dnf history rollback** コマンドを使用して、指定されたトランザクションと最後のトランザクションの間に実行されたすべての DNF トランザクションを元に戻します。



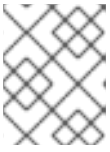
重要

dnf history undo コマンドと **dnf history rollback** コマンドを使用して RHEL システムパッケージを古いバージョンにダウングレードすることはサポートされていません。これは特に、**selinux**、**selinux-policy-***、**kernel**、および **glibc** パッケージ、ならびに **gcc** などの **glibc** の依存関係に関係します。したがって、システムをマイナーバージョンにダウングレードすると (たとえば、RHEL 9.1 から RHEL 9.0 に)、システムが不正な状態になる可能性があるため、推奨されません。

9.2.1. 単一の DNF トランザクションを元に戻す

dnf history undo コマンドを使用して、単一のトランザクション内で実行されたステップを元に戻すことができます。

- このトランザクションで新しいパッケージがインストールされた場合は、**dnf history undo** がそのパッケージをアンインストールします。
- トランザクションでパッケージがアンインストールされた場合は、**dnf history undo** がそのパッケージを再インストールします。
- また、**dnf history undo** コマンドは、古いパッケージが依然として利用可能な場合、更新されたすべてのパッケージを以前のバージョンにダウングレードしようとします。



注記

古いパッケージバージョンが利用できない場合は、**dnf history undo** コマンドを使用したダウングレードは失敗します。

手順

1. 元に戻すトランザクションの ID を特定します。

```
# 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. オプション: 詳細を表示して、元に戻したいトランザクションであることを確認します。

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

3. トランザクションを元に戻します。

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

たとえば、以前にインストールした **unzip** パッケージをアンインストールする場合は、次のように入力します。

```
# dnf history undo 12
```

9.2.2. 複数の DNF トランザクションを元に戻す

dnf history rollback コマンドを使用して、指定したトランザクションと最後のトランザクションの間に実行されたすべての **DNF** トランザクションを元に戻すことができます。トランザクション ID で指定されたトランザクションは変更されないことに注意してください。

手順

1. 元に戻したい状態のトランザクション ID を特定します。

```
# 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. 指定したトランザクションを元に戻す。

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

たとえば、**wget** および **unzip** パッケージがインストールされる前の状態に戻すには、次のように入力します。

```
# dnf history rollback 12
```

または、トランザクション履歴のすべてのトランザクションを元に戻すには、トランザクション ID **1** を使用します。

```
# dnf history rollback 1
```

第10章 カスタムソフトウェアリポジトリーの管理

/etc/dnf/dnf.conf ファイルまたは **/etc/yum.repos.d/** ディレクトリーの **.repo** ファイルでリポジトリーを設定できます。

重要

/etc/dnf/dnf.conf ではなく **.repo** ファイルでリポジトリーを定義します。

/etc/dnf/dnf.conf ファイルには **[main]** セクションが含まれており、リポジトリー固有のオプションを設定するために使用できる 1 つ以上のリポジトリーセクション (**[<repository-ID>]**) を含めることができます。**/etc/dnf/dnf.conf** ファイルの個々のリポジトリーセクションで定義した値は、**[main]** セクションで設定されたオーバーライド値です。

10.1. DNF リポジトリーオプション

/etc/dnf/dnf.conf 設定ファイルには、括弧 (**[]**) で囲まれた一意のリポジトリー ID を持つリポジトリーセクションが含まれています。このようなセクションを使用して、個々の DNF リポジトリーを定義できます。

重要

[] 内のリポジトリー ID は一意である必要があります。

使用可能なリポジトリー ID オプションの完全なリストについては、**dnf.conf(5)** の man ページの **[<repository-ID>] OPTIONS** セクションを参照してください。

10.2. DNF リポジトリーの追加

dnf config-manager --add-repo コマンドを使用して、DNF リポジトリーをシステムに追加できます。

手順

1. システムにリポジトリーを追加します。

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

   このコマンドによって追加されたリポジトリーはデフォルトで有効になっていることに注意してください。

2. 前のコマンドで **/etc/yum.repos.d/<repository_URL>.repo** ファイルに作成されたリポジトリー設定を確認し、必要に応じて更新します。

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

警告

ソフトウェアパッケージを、Red Hat の認証ベース **Content Delivery Network (CDN)** 以外の未検証または信頼できないソースから取得してインストールする場合には、セキュリティー上のリスクが伴います。セキュリティー、安定性、互換性、保全性に関する問題につながる恐れがあります。

10.3. DNF リポジトリーの有効化

dnf config-manager コマンドを使用して、システムに追加された DNF リポジトリーを有効にすることができます。

手順

- リポジトリーを有効にします。

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

10.4. DNF リポジトリーの無効化

dnf config-manager コマンドを使用して、システムに追加された DNF リポジトリーを無効にすることができます。

手順

- リポジトリーを無効にします。

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

第11章 APPLICATION STREAMS のコンテンツのバージョンの管理

AppStream リポジトリーのコンテンツは、モジュールストリームに対応する複数のバージョンが使用できます。

11.1. モジュールの依存関係とストリームの変更

これまで、コンテンツを提供するパッケージは他のパッケージに依存し、通常は使用する依存関係バージョンを指定していました。モジュールに含まれるパッケージにもこの仕組みが適用されますが、パッケージとその特定バージョンをモジュールとストリームとしてグループ化したことで、さらに制限されます。また、モジュールストリームは含まれるパッケージや提供するパッケージに制限されずに、その他のモジュールのストリームへの依存関係を宣言できます。

パッケージやモジュールを操作した後は、インストールされているすべてのパッケージの依存関係ツリー全体が、パッケージが宣言しているすべての条件を満たさなければなりません。また、すべてのモジュールストリームの依存関係に適合する必要があります。あるモジュールストリームを無効にすると、他のモジュールストリームも無効にしなければならない場合があります。自動的に削除されるパッケージはありません。

次のアクションにより、後続の自動操作が発生する可能性があることに注意してください。

- モジュールストリームを有効にすると、別のモジュールストリームが有効になる可能性があります。
- モジュールストリームプロファイルをインストールしたり、ストリームからパッケージをインストールしたりすると、他のモジュールストリームが有効になり、他のパッケージがインストールされる可能性があります。
- パッケージを削除すると、別のパッケージも削除される可能性があります。このようなパッケージがモジュールにより提供されている場合は、このモジュールストリームのパッケージがインストールされなくなっても、将来のインストールに備えてモジュールストリームは有効のままになります。これは、未使用の **DNF** リポジトリーの挙動を反映しています。

11.2. モジュールおよび非モジュールの依存関係のやり取り

モジュールの依存関係 は、通常の RPM 依存関係の上の追加レイヤーです。モジュール依存関係の機能は、リポジトリー間で仮想的な依存関係と同様になります。つまり、異なるパッケージをインストールするには、RPM の依存関係と、モジュールの依存関係の両方を解決する必要があります。

変更が明示的に指示される場合を除き、システムは常にモジュールとストリームの選択を保持します。モジュールパッケージは、このパッケージを提供するモジュールの、現在有効なストリームに含まれる更新を受け取りますが、別のストリームに含まれたバージョンへのアップグレードは行いません。

11.3. モジュールストリームのリセット

モジュールをリセットすると、このモジュールが有効でも無効でもない初期状態に戻ります。モジュールに設定したデフォルトストリームがある場合は、モジュールをリセットするとそのストリームがアクティブになります。

モジュールをリセットすると、たとえば、モジュールを有効にしたままにせずに、モジュールから RPM コンテンツのみを抽出したい場合に便利です。モジュールを有効にしてその内容を抽出した後、**dnf module reset** コマンドを使用してこのモジュールを初期状態にリセットできます。

手順

- モジュールストリームをリセットにします。

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

モジュールは初期状態に戻ります。有効なストリームとインストールされたプロファイルに関する情報は消去されますが、インストールされたコンテンツは削除されません。

11.4. モジュールのストリームをすべて無効化

デフォルトストリームを持つモジュールは、常に 1 つのストリームがアクティブになります。モジュールのすべてのモジュールストリームのコンテンツにアクセスできないようにする場合は、モジュール全体を無効にできます。

前提条件

- アクティブモジュールストリーム の概念 を理解している。

手順

- モジュールを無効にします。

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

dnf コマンドにより確認が求められた後に、モジュールとそのすべてのストリームが無効になります。すべてのモジュールが非アクティブになります。インストール済みのコンテンツは削除されません。

11.5. 後のストリームへの切り替え

後のモジュールストリームに切り替えると、それぞれのパッケージがすべて後のバージョンに置き換えられます。

重要

データをバックアップし、コンポーネントに固有の移行手順に従ってください。

前提条件

- システムが完全に更新されている。

手順

1. インストールされているコンポーネントを新しいバージョンに切り替え、モジュール (コンポーネント) とストリーム (バージョン) を選択します。

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

たとえば、**nodejs:18** モジュールストリームから **nodejs:20** ストリームに切り替えるには、次のように実行します。

```
# 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
```

非モジュールコンテンツからモジュールストリームに切り替えることもできます。たとえば、非モジュラー **PHP 8.0** からモジュラー **PHP 8.1**に切り替えるには、次のように実行します。

```
# 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. オプション: インストールされているコンポーネントを新しいバージョンに切り替えます。インストールまたは更新するプロファイルも選択します。

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

検証

- インストールされたコンポーネントが新しいバージョンに切り替わったことを確認します (**[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
```

付録A DNF コマンドリスト

以下のセクションでは、Red Hat Enterprise Linux 9 のコンテンツを一覧表示、インストール、および削除するための DNF コマンドを説明します。

A.1. RHEL 9 でコンテンツをリスト表示するコマンド

以下は、Red Hat Enterprise Linux 9 でコンテンツとその詳細を検索するために一般的に使用される DNF コマンドです。

コマンド	説明
dnf search term	パッケージに関連する用語を使用してパッケージを検索します。
dnf repoquery package	選択したパッケージとそのバージョンの有効な DNF リポジトリーを検索します。
dnf list	インストール済みおよび利用可能なすべてのパッケージに関する情報を一覧表示します。
dnf list --installed **dnf repoquery --installed**	システムにインストールされているパッケージを一覧表示します。
dnf list --available **dnf repoquery**	インストール可能な有効なリポジトリーにある、すべてのパッケージを一覧表示します。
dnf repolist	システムで有効なリポジトリーを一覧表示します。
dnf repolist --disabled	システムで無効になっているリポジトリーを一覧表示します。
dnf repolist --all	有効および無効の両方のリポジトリーを一覧表示します。
dnf repoinfo	リポジトリーの追加情報を一覧表示します。
dnf info package_name **dnf repoquery --info package_name**	利用可能なパッケージの詳細を表示します。
dnf repoquery --info --installed package_name	システムにインストールされているパッケージの詳細を表示します。
dnf module list	モジュールとその現在の状態を一覧表示します。
dnf module info module_name	モジュールの詳細を表示します。

コマンド	説明
dnf module list module_name	モジュールの現在の状態を表示します。
dnf module info --profile module_name	選択したモジュールの利用可能なプロファイルに関連付けられたパッケージを表示します。
dnf module info --profile module_name:stream	指定されたストリームを使用して、モジュールの利用可能なプロファイルに関連付けられたパッケージを表示します。
dnf module provides package	パッケージを提供するモジュール、ストリーム、およびプロファイルを特定します。 なお、パッケージがどのモジュールからも利用できない場合は、このコマンドの出力は空になります。
dnf group summary	インストールされているグループと利用可能なグループの数を表示します。
dnf group list	インストール済みおよび利用可能なグループを一覧表示します。
dnf group info group_name	特定のグループに含まれる必須および任意のパッケージを一覧表示します。

A.2. RHEL 9 にコンテンツをインストールするコマンド

以下は、Red Hat Enterprise Linux 9 にコンテンツをインストールするために一般的に使用される **DNF** コマンドです。

コマンド	説明
dnf install package_name	パッケージのインストール モジュールストリームによりパッケージが提供される場合は、**dnf** が必要なモジュールストリームを解決し、このパッケージのインストール時に自動的に有効になります。これは、すべてのパッケージ依存関係で再帰的に行われます。より多くのモジュールストリームがこの要件を満たす必要がある場合は、デフォルトのモジュールストリームが使用されます。
dnf install package_name_1 package_name_2	複数のパッケージとその依存関係を同時にインストールします。

コマンド	説明
dnf install package_name.arch	**multilib** システム (AMD64、Intel 64 マシン) にパッケージをインストールするときに、パッケージ名に追加してパッケージのアーキテクチャーを指定します。
dnf install /usr/sbin/binary_file	バイナリーへのパスを引数として使用して、バイナリーをインストールします。
dnf install /path/	ダウンロード済みのパッケージをローカルディレクトリーからインストールします。
dnf install package_url	パッケージ URL を使用してリモートパッケージをインストールします。
dnf module enable module_name:stream	特定のストリームを使用してモジュールを有効にします。 このコマンドを実行しても RPM パッケージはインストールされないことに注意してください。
dnf module install module_name:stream **dnf install @module_name:stream**	特定のモジュールストリームからデフォルトのプロファイルをインストールします。 このコマンドを実行すると、指定したストリームも有効になっていることに注意してください。
dnf module install module_name:stream/profile **dnf install @module_name:stream/profile**	特定のストリームを使用して、選択したプロファイルをインストールします。
dnf group install group_name	グループ名でパッケージグループをインストールします。
dnf group install group_ID	groupID でパッケージグループをインストールします。

A.3. RHEL 9 でコンテンツを削除するコマンド

以下は、Red Hat Enterprise Linux 9 でコンテンツを削除するために一般的に使用される **DNF** コマンドです。

コマンド	説明
dnf remove package_name	特定のパッケージとすべての依存パッケージを削除します。

コマンド	説明
dnf remove package_name_1 package_name_2	複数のパッケージと、その未使用の依存関係を同時に削除します。
dnf group remove group_name	グループ名でパッケージグループを削除します。
dnf group remove group_ID	groupID でパッケージグループを削除します。
dnf module remove --all module_name:stream	指定されたストリームからすべてのパッケージを削除します。 このコマンドを実行すると、システムから重要なパッケージが削除される可能性があることに注意してください。
dnf module remove module_name:stream/profile	インストール済みプロファイルからのパッケージを削除します。
dnf module remove module_name:stream	指定したストリーム内のすべてのインストール済みプロファイルからのパッケージを削除します。
dnf module reset module_name	モジュールを初期状態へのリセット このコマンドを実行しても、指定したモジュールからパッケージが削除されないことに注意してください。
dnf module disable module_name	モジュールおよびそのストリームの無効化 このコマンドを実行しても、指定したモジュールからパッケージが削除されないことに注意してください。