Red Hat Developer Tools 1

Rust 1.62.1 Toolset の使用

Rust 1.62.1 Toolset のインストールおよび使用

Last Updated: 2023-06-29

Red Hat Developer Tools 1 Rust 1.62.1 Toolset の使用

Rust 1.62.1 Toolset のインストールおよび使用

Jacob Valdez
jvaldez@redhat.com

Eva-Lotte Gebhardt
egebhard@redhat.com

Zuzana Zoubkova

Peter Macko

Kevin Owen

Vladimir Slavik

法律上の通知

概要

Rust Toolset は、Red Hat Enterprise Linux (RHEL) オペレーティングシステム上の開発者向けの Red Hat 製品です。本書では、Rust Toolset の概要、さまざまなバージョンの Rust ツールを起動して使用する方法、および詳細な情報を含むリソースを検索する方法を説明します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 RUST TOOLSET

Rust Toolset は、Red Hat Enterprise Linux (RHEL) 上の開発者向けの Red Hat 製品です。Rust プログラミング言語、Rust パッケージマネージャー Cargo、**rustfmt** フォーマットツール、および必要なライブラリーに **rustc** コンパイラーを提供します。

Rust Toolset は、Red Hat Developer Tools for Red Hat Enterprise Linux 7 の一部として配布されています。Red Hat Enterprise Linux 8 では、Rust Toolset はモジュールとして利用できます。Rust Toolset は、Red Hat Enterprise Linux 9 のパッケージとして利用できます。

1.1. RUST TOOLSET コンポーネント

以下のコンポーネントは、Rust Toolset の一部として利用できます。

Name	バージョン	説明
rust	1.62.1	LLVM 用の Rust コンパイラーフロントエンド。
cargo	1.62.1	Rust のビルドシステムおよび依存関係マネージャー。
rustfmt	1.62.1	Rust コードの自動フォーマットを行うためのツール。

1.2. RUST TOOLSET の互換性

Rust Toolset は、以下のアーキテクチャーで Red Hat Enterprise Linux 7、Red Hat Enterprise Linux 8、および Red Hat Enterprise Linux 9 で利用できます。

- AMD および Intel 64 ビット
- 64 ビット ARM (RHEL 8 および RHEL 9)
- IBM Power Systems、リトルエンディアン
- IBM Power Systems、ビックエンディアン (RHEL 7 のみ)
- 64 ビット IBM Z

1.3. RED HAT ENTERPRISE LINUX 7 での RUST TOOLSET へのアクセス

Rust Toolset を Red Hat Enterprise Linux 7 にインストールするには、Red Hat Developer Tools リポジトリーおよび Red Hat Software Collections リポジトリーにアクセスして有効にする必要があります。これらのリポジトリーがすでにシステムに割り当てられている場合は、Rust Toolset のインストール を参照してください。

手順

1. 以下を実行して **Wget** をインストールします。

```
# yum install wget
```

2. 以下を実行して最新のサブスクリプションデータをダウンロードします。

```
# subscription-manager refresh
```

3. 以下を実行してシステムを登録します。

```
# subscription-manager register
```

グラフィカルユーザーインターフェイス (GUI) を使用してシステムを登録するには、システムの登録と登録解除 ガイドを参照してください。

4. 次のコマンドを実行して、利用可能なすべてのサブスクリプションのリストを表示し、プール ID を特定します。

```
# subscription-manager list --available
```

5. **Pool ID** で始まる行でプール ID を検索します。

6. 以下を実行して、**Red Hat Developer Tools** リポジトリーへのアクセスを提供するサブスクリプションをシステムに割り当てます。

```
# subscription-manager attach --pool=<pool ID from the subscription>
```

- **<pool ID from the subscription>** は、直前の手順で特定したプール ID に置き換えます。

7. 以下を実行して、システムに割り当てられているサブスクリプションを確認します。

```
# sudo subscription-manager list --consumed
```

8. 以下を実行して **rhel-7-variant-devtools-rpms** リポジトリーを有効にします。

```
# subscription-manager repos --enable rhel-7-<variant>-devtools-rpms
```

- **<variant>** は、Red Hat Enterprise Linux システムのバリアント (**server** または **workstation**) に置き換えます。
  最も幅広い開発ツールにアクセスするには、**server** を使用します。

9. 以下を実行して **rhel-variant-rhscl-7-rpms** リポジトリーを有効にします。

```
# subscription-manager repos --enable rhel-<variant>-rhscl-7-rpms
```

- **<variant>** は、Red Hat Enterprise Linux システムのバリアント (**server** または **workstation**) に置き換えます。

10. 以下を実行して Red Hat Developer Tools GPG キーをシステムに追加します。

```
# cd /etc/pki/rpm-gpg
# wget -O RPM-GPG-KEY-redhat-devel https://www.redhat.com/security/data/a5787476.txt
# rpm --import RPM-GPG-KEY-redhat-devel
```

関連情報

- システムを登録し、サブスクリプションに関連付ける方法は、Red Hat Subscription Management ガイドコレクションを参照してください。

1.4. RUST TOOLSET のインストール

すべての開発ツール、デバッグツール、および依存パッケージを含む Rust Toolset をインストールするには、以下の手順を実行します。Rust Toolset には、LLVM Toolset の依存関係があることに注意してください。

前提条件

- Red Hat Enterprise Linux 7 で、Red Hat Developer Tools コンテンツセットへのアクセスを提供するサブスクリプションがシステムに割り当てられている。
  サブスクリプションを割り当てるには、Red Hat Enterprise Linux 7 での Rust Toolset へのアクセス を参照してください。

- 利用可能な Red Hat Enterprise Linux のすべての更新がインストールされている。

手順

Red Hat Enterprise Linux 7 に、**rust-toolset-1.62** コレクションをインストールします。

```
# yum install rust-toolset-1.62
```

Red Hat Enterprise Linux 8 で、以下のコマンドを実行して **rust-toolset** モジュールをインストールします。

```
# yum module install rust-toolset
```

Red Hat Enterprise Linux 9 で、以下のコマンドを実行して **rust-toolset** パッケージをインストールします。

```
# dnf install rust-toolset
```

1.5. RUST ドキュメントのインストール

The Rust Programming Language ガイドは、インストール可能なドキュメントとして利用できます。

前提条件

- Rust Toolset がインストールされている。
  詳細は、Rust Toolset のインストール を参照してください。

手順

rust-doc パッケージをインストールするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 7 の場合

  ```
  # yum install rust-toolset-1.62-rust-doc
  ```

  Rust プログラミング言語 のガイドは **/opt/rh/rust-toolset-1.62/root/usr/share/doc/rust/html/index.html** にあります。
  すべての Rust コードパッケージの API ドキュメントは、**/opt/rh/rust-toolset-**

1.62/root/usr/share/doc/rust/html/std/index.html にあります。

- Red Hat Enterprise Linux 8 の場合

```
# yum install rust-doc
```

The Rust Programming Language ガイドは、**/usr/share/doc/rust/html/index.html** にあります。
すべての Rust コードパッケージの API ドキュメントは、**/usr/share/doc/rust/html/std/index.html** にあります。

- Red Hat Enterprise Linux 9 の場合

```
# dnf install rust-doc
```

The Rust Programming Language ガイドは、**/usr/share/doc/rust/html/index.html** にあります。
すべての Rust コードパッケージの API ドキュメントは、**/usr/share/doc/rust/html/std/index.html** にあります。

1.6. CARGO ドキュメントのインストール

Cargo (Rust のパッケージマネージャー) ガイドは、Cargo のインストール可能なドキュメントとして利用できます。

前提条件

- Rust Toolset がインストールされている。
詳細は、Rust Toolset のインストール を参照してください。

手順

- **cargo-doc** パッケージをインストールするには、以下のコマンドを実行します。
  - Red Hat Enterprise Linux 7 の場合

```
# yum install rust-toolset-1.62-cargo-doc
```

Cargo book は、**/opt/rh/rust-toolset-1.62/root/usr/share/doc/cargo/html/index.html** にあります。

  - Red Hat Enterprise Linux 8 の場合

```
# yum install cargo-doc
```

Cargo (Rust のパッケージマネージャー) のガイドは、**/usr/share/doc/cargo/html/index.html** にあります。

  - Red Hat Enterprise Linux 9 の場合

```
# dnf install cargo-doc
```

Cargo (Rust のパッケージマネージャー) のガイドは、**/usr/share/doc/cargo/html/index.html** にあります。

1.7. 関連情報

- Rust プログラミング言語の詳細は、Rust の公式ドキュメント を参照してください。

第2章 CARGO ビルドツール

Cargo は、Rust コンパイラー **rustc** と、パッケージおよび依存関係マネージャーのビルドツールおよびフロントエンドです。これにより、Rust プロジェクトは、特定のバージョン要件での依存関係の宣言、完全な依存関係グラフの解決、パッケージのダウンロード、プロジェクト全体のビルドとテストが可能となります。

Rust Toolset には、Cargo 1.62.1 が同梱されています。

2.1. CARGO ディレクトリー構造およびファイルの配置

Cargo ビルドツールは、セット規則を使用して Cargo パッケージ内のディレクトリー構造とファイル配置を定義します。**cargo new** コマンドを実行すると、マニフェストとプロジェクトファイル両方のパッケージディレクトリー構造とテンプレートが生成されます。デフォルトでは、パッケージの root ディレクトリーで新しい Git リポジトリーも初期化します。

バイナリープログラムの場合、Cargo は、**Cargo.toml** という名前のテキストファイルを含むディレクトリー **project_name** と、**main.rs** という名前のテキストファイルを含むサブディレクトリー **src** を作成します。

関連情報

- Cargo ディレクトリー構造の詳細は、The Cargo Book – Package Layout を参照してください。
- Rust コードの整理に関する詳細は、The Rust Programming Language – Managing Growing Projects with Packages, Crates, and Modules を参照してください。

2.2. RUST プロジェクトの作成

Cargo の規則に従って設定される新しい Rust プロジェクトを作成します。Cargo の規則に関する詳細は、Cargo ディレクトリー構造およびファイルの配置 を参照してください。

手順

以下のコマンドを実行して Rust プロジェクトを作成します。

- Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo new --bin <project_name>'
```

  - **<project_name>** をプロジェクト名に置き換えます。

- Red Hat Enterprise Linux 8 の場合

```
$ cargo new --bin <project_name>
```

  - **<project_name>** をプロジェクト名に置き換えます。

- Red Hat Enterprise Linux 9 の場合

```
$ cargo new --bin <project_name>
```

  - **<project_name>** をプロジェクト名に置き換えます。

注記

プロジェクトコードを編集するには、メインの実行可能ファイル **main.rs** を編集し、**src** サブディレクトリーに新しいソースファイルを追加します。

関連情報

- プロジェクトの設定および依存関係の追加に関する詳細は、Rust プロジェクト依存関係の設定 を参照してください。

2.3. RUST プロジェクトのライブラリーの作成

Cargo ビルドツールを使用して Rust プロジェクトのライブラリーを作成するには、以下の手順を実行します。

前提条件

- 既存の Rust プロジェクトがある。
  Rust プロジェクトの作成方法は、Rust プロジェクトの作成 を参照してください。

手順

Rust プロジェクトのライブラリーを作成するには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo new --lib <project_name>'
```

  - **<project_name>** を Rust プロジェクトの名前に置き換えます。

- Red Hat Enterprise Linux 8 の場合

```
$ cargo new --lib <project_name>
```

  - **<project_name>** を Rust プロジェクトの名前に置き換えます。

- Red Hat Enterprise Linux 9 の場合

```
$ cargo new --lib <project_name>
```

  - **<project_name>** を Rust プロジェクトの名前に置き換えます。

2.4. RUST プロジェクトのビルド

Cargo ビルドツールを使用して Rust プロジェクトをビルドします。Cargo は、プロジェクトのすべての依存関係を解決し、不足している依存関係をダウンロードして、**rustc** コンパイラーを使用してコンパイルします。

デフォルトでは、プロジェクトはデバッグモードでビルドおよびコンパイルされます。リリースモードでのプロジェクトのコンパイルに関する情報は、リリースモードでの Rust プロジェクトのビルド を参照してください。

前提条件

- 既存の Rust プロジェクトがある。
  Rust プロジェクトの作成方法は、Rust プロジェクトの作成 を参照してください。

手順

- Cargo によって管理される Rust プロジェクトをビルドするには、プロジェクトディレクトリーで実行します。
  - Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo build'
```

  - Red Hat Enterprise Linux 8 の場合

```
$ cargo build
```

  - Red Hat Enterprise Linux 9 の場合

```
$ cargo build
```

- 実行ファイルをビルドする必要がない場合に Rust プログラムをビルドするには、以下を実行します。

```
$ cargo check
```

2.5. リリースモードでの RUST プロジェクトのビルド

Cargo ビルドツールを使用して、リリースモードで Rust プロジェクトをビルドします。リリースモードはソースコードの最適化を行うため、コンパイルされたバイナリーの実行速度が速くなる一方で、コンパイル時間を増やすことができます。このモードを使用して、リリースと実稼働環境に適した最適化されたアーティファクトを生成します。
Cargo は、プロジェクトのすべての依存関係を解決し、不足している依存関係をダウンロードして、**rustc** コンパイラーを使用してコンパイルします。

デバッグモードでプロジェクトをコンパイルする方法は、Rust プロジェクトのビルド を参照してください。

前提条件

- 既存の Rust プロジェクトがある。
  Rust プロジェクトの作成方法は、Rust プロジェクトの作成 を参照してください。

手順

- リリースモードでプロジェクトをビルドするには、以下を実行します。
  - Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo build --release'
```

  - Red Hat Enterprise Linux 8 の場合

```
$ cargo build --release
```

- Red Hat Enterprise Linux 9 の場合

```
$ cargo build --release
```

- 実行ファイルをビルドする必要がない場合に Rust プログラムをビルドするには、以下を実行します。

```
$ cargo check
```

2.6. RUST プログラムの実行

Cargo ビルドツールを使用して Rust プロジェクトを実行します。Cargo はまずプロジェクトを再ビルドし、作成された実行ファイルを実行します。開発中に使用される場合、**cargo run** コマンドはビルドモードとは独立して出力パスを正しく解決します。

前提条件

- ビルド済みの Rust プロジェクトがある。
  Rust プロジェクトのビルド方法は、Rust プロジェクトのビルド を参照してください。

手順

Cargo によってプロジェクトとして管理される Rust プログラムを実行するには、プロジェクトディレクトリーで実行します。

- Red Hat Enterprise Linux 7 の場合

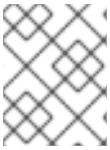
```
$ scl enable rust-toolset-1.62 'cargo run'
```

- Red Hat Enterprise Linux 8 の場合

```
$ cargo run
```

- Red Hat Enterprise Linux 9 の場合

```
$ cargo run
```



注記

プログラムがビルドされていない場合、Cargo は実行する前にプログラムをビルドします。

2.7. RUST プロジェクトのテスト

Cargo ビルドツールを使用して Rust プログラムをテストします。Cargo はまずプロジェクトを再ビルドし、次にプロジェクトにあるテストを実行します。テストできるのは、自由で単相的で引数を取らない関数のみであることに注意してください。関数の戻り値のタイプは **()** または **Result<(), E> where E: Error** のいずれかでなければなりません。

デフォルトでは、Rust プロジェクトはデバッグモードでテストされています。リリースモードでプロジェクトをテストする方法は、リリースモードでの Rust プロジェクトのテスト を参照してください。

前提条件

前提条件

- ビルド済みの Rust プロジェクトがある。
  Rust プロジェクトのビルド方法は、Rust プロジェクトのビルド を参照してください。

手順

- 関数の前にテスト属性 **#[test]** を追加します。
- Cargo が管理する Rust プロジェクトのテストを実行するには、プロジェクトディレクトリーで実行します。
  - Red Hat Enterprise Linux 7 の場合

    ```
    $ scl enable rust-toolset-1.62 'cargo test'
    ```

  - Red Hat Enterprise Linux 8 の場合

    ```
    $ cargo test
    ```

  - Red Hat Enterprise Linux 9 の場合

    ```
    $ cargo test
    ```

関連情報

- Rust プロジェクトでテストを実行する方法は、The Rust Reference – Testing attributes を参照してください。

2.8. リリースモードでの RUST プロジェクトのテスト

Cargo ビルドツールを使用して、リリースモードで Rust プログラムをテストします。リリースモードはソースコードの最適化を行うため、コンパイルされたバイナリーの実行速度が速くなる一方で、コンパイル時間を増やすことができます。このモードを使用して、リリースと実稼働環境に適した最適化されたアーティファクトを生成します。
Cargo はまずプロジェクトを再ビルドし、次にプロジェクトにあるテストを実行します。テストできるのは、自由で単相的で引数を取らない関数のみであることに注意してください。関数の戻り値のタイプは **()** または **Result<(), E> where E: Error** のいずれかでなければなりません。

デバッグモードでプロジェクトをテストする方法は、Rust プロジェクトのテスト を参照してください。

前提条件

- ビルド済みの Rust プロジェクトがある。
  Rust プロジェクトのビルド方法は、Rust プロジェクトのビルド を参照してください。

手順

- 関数の前にテスト属性 **#[test]** を追加します。
- Cargo が管理する Rust プロジェクトのテストをリリースモードで実行するには、プロジェクトディレクトリーで実行します。
  - Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo test --release'
```

- Red Hat Enterprise Linux 8 の場合

```
$ cargo test --release
```

- Red Hat Enterprise Linux 9 の場合

```
$ cargo test --release
```

関連情報

- Rust プロジェクトでテストを実行する方法は、The Rust Reference – Testing attributes を参照してください。

2.9. RUST プロジェクト依存関係の設定

Cargo ビルドツールを使用して Rust プロジェクトの依存関係を設定します。Cargo が管理するプロジェクトの依存関係を指定するには、プロジェクトディレクトリーの **Cargo.toml** ファイルを編集して、プロジェクトを再ビルドします。Cargo は、Rust コードパッケージとその依存関係のダウンロード、それらのローカルへの保存、依存関係コードパッケージを含むすべてのプロジェクトソースコードのビルド、生成される実行ファイルの実行を行います。

前提条件

- ビルド済みの Rust プロジェクトがある。
  Rust プロジェクトのビルド方法は、Rust プロジェクトのビルド を参照してください。

手順

1. プロジェクトディレクトリーで **Cargo.toml** ファイルを開きます。
2. セクションラベル **[dependencies]** に移動します。
   各依存関係は、次の形式で新しい行に一覧表示されます。

   crate_name = **version**

   Rust コードパッケージは crates と呼ばれます。
3. 依存関係を編集します。
4. 以下を実行してプロジェクトを再ビルドします。
   - Red Hat Enterprise Linux 7 の場合

     ```
     $ scl enable rust-toolset-1.62 'cargo build'
     ```

   - Red Hat Enterprise Linux 8 の場合

     ```
     $ cargo build
     ```

   - Red Hat Enterprise Linux 9 の場合

```
$ cargo build
```

5. 以下のコマンドを使用してプロジェクトを実行します。

- Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo run'
```

- Red Hat Enterprise Linux 8 の場合

```
$ cargo run
```

- Red Hat Enterprise Linux 9 の場合

```
$ cargo run
```

関連情報

- Rust 依存関係の設定に関する詳細は、The Cargo Book – Specifying Dependencies を参照してください。

2.10. RUST プロジェクトのドキュメントのビルド

Cargo ツールを使用して、抽出用にマークされたソースコードのコメントからドキュメントを生成します。ドキュメントのコメントは、パブリック関数、変数、メンバーに対してのみ抽出されることに注意してください。

前提条件

- ビルド済みの Rust プロジェクトがある。
  Rust プロジェクトのビルド方法は、Rust プロジェクトのビルド を参照してください。
- 設定された依存関係がある。
  依存関係の設定に関する詳細は、Rust プロジェクト依存関係の設定 を参照してください。

手順

- 抽出用のコメントをマークするには、3 つのスラッシュ /// を使用して、ドキュメント化された行頭にコメントを配置します。
  Cargo は、コメントのマークダウン言語をサポートしています。
- Cargo を使用してプロジェクトのドキュメントをビルドするには、プロジェクトディレクトリーで実行します。
  - Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo doc --no-deps'
```

  - Red Hat Enterprise Linux 8 の場合

```
$ cargo doc --no-deps
```

  - Red Hat Enterprise Linux 9 の場合

```
$ cargo doc --no-deps
```

生成されたドキュメントは **.target/doc** ディレクトリーにあります。

関連情報

- Cargo を使用したドキュメントのビルドに関する詳細は、The Rust Programming Language – Making Useful Documentation Comments を参照してください。

2.11. RED HAT ENTERPRISE LINUX 8 および RED HAT ENTERPRISE LINUX 9 ベータ版で RUST を使用してコードを WEBASSEMBLY バイナリーにコンパイルする

WebAssembly 標準ライブラリーをインストールするには、以下の手順を実行します。

前提条件

- Rust Toolset がインストールされている。
  詳細は、Rust Toolset のインストール を参照してください。

手順

- WebAssembly 標準ライブラリーをインストールするには、以下を実行します。
  - Red Hat Enterprise Linux 8 の場合

    ```
    # yum install rust-std-static-wasm32-unknown-unknown
    ```

  - Red Hat Enterprise Linux 9 の場合

    ```
    # dnf install rust-std-static-wasm32-unknown-unknown
    ```

- Cargo で WebAssembly を使用するには、以下を実行します。
  - Red Hat Enterprise Linux 8 の場合

    ```
    # cargo <command> --target wasm32-unknown-unknown
    ```

    <**command**> を、実行する Cargo コマンドに置き換えます。

  - Red Hat Enterprise Linux 9 の場合

    ```
    # cargo <command> --target wasm32-unknown-unknown
    ```

    <**command**> を、実行する Cargo コマンドに置き換えます。

関連情報

- WebAssembly の詳細は、Rust および WebAssemblyの公式ドキュメント、または Rust および WebAssembly ガイドを参照してください。

2.12. RUST プロジェクトの依存関係のベンダー化

オフライン再分配用に、Rust プロジェクトの依存関係のローカルコピーを作成し、Cargo ビルドツールを使用して再利用します。この手順は、プロジェクトの依存関係のベンダー化と呼ばれます。Windows オペレーティングシステムでプロジェクトをビルドする Rust コードパッケージを含むベンダー化された依存関係は、**vendor** ディレクトリーにあります。ベンダー化された依存関係は、インターネットに接続せずに Cargo で使用できます。

前提条件

- ビルド済みの Rust プロジェクトがある。
  Rust プロジェクトのビルド方法は、Rust プロジェクトのビルド を参照してください。
- 設定された依存関係がある。
  依存関係の設定に関する詳細は、Rust プロジェクト依存関係の設定 を参照してください。

手順

Cargo を使用して依存関係を含む Rust プロジェクトをベンダー化するには、プロジェクトディレクトリーで以下を実行します。

- Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo vendor'
```

- Red Hat Enterprise Linux 8 の場合

```
$ cargo vendor
```

- Red Hat Enterprise Linux 9 の場合

```
$ cargo vendor
```

2.13. 関連情報

- Cargo の詳細は、Cargo の公式ガイド を参照してください。
- Rust Toolset に含まれる man ページを表示するには、以下を実行します。
  - Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'man cargo'
```

  - Red Hat Enterprise Linux 8 の場合

```
$ man cargo
```

  - Red Hat Enterprise Linux 9 の場合

```
$ man cargo
```

第3章 RUSTFMT フォーマットツール

rustfmt フォーマットツールで、Rust プログラムのソースコードを自動的にフォーマットできます。**rusftmt** は、スタンドアロンツールまたは Cargo で使用できます。

3.1. RUSTFMT のインストール

rustfmt フォーマットツールをインストールするには、以下の手順を実行します。

前提条件

- Rust Toolset がインストールされている。
  詳細は、Rust Toolset のインストール を参照してください。

手順

以下のコマンドを実行して **rustfmt** をインストールします。

- Red Hat Enterprise Linux 7 の場合

```
# yum install rust-toolset-1.62-rustfmt
```

- Red Hat Enterprise Linux 8 の場合

```
# yum install rustfmt
```

- Red Hat Enterprise Linux 9 の場合

```
# dnf install rustfmt
```

3.2. スタンドアロンツールとしての RUSTFMT の使用

rustfmt をスタンドアロンツールとして使用し、Rust ソースファイルとそのすべての依存関係をフォーマットします。別の方法として、Cargo ビルドツールで **rustfmt** を使用します。詳細は、Cargo ビルドツールでの rustfmt の使用 を参照してください。

前提条件

- 既存の Rust プロジェクトがある。
  Rust プロジェクトの作成方法は、Rust プロジェクトの作成 を参照してください。

手順

rustfmt をスタンドアロンツールとして使用して Rust ソースファイルをフォーマットするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'rustfmt <source-file>'
```

  - **<source_file>** をソースファイルの名前に置き換えます。
    <source_file> を標準入力に置き換えることもできます。次に、**rustfmt** は標準出力に出力を提供します。

- Red Hat Enterprise Linux 8 の場合

```
$ rustfmt <source-file>
```

  - **<source_file>** をソースファイルの名前に置き換えます。
    <source_file> を標準入力に置き換えることもできます。次に、**rustfmt** は標準出力に出力を提供します。

- Red Hat Enterprise Linux 9 の場合

```
$ rustfmt <source-file>
```

  - **<source_file>** をソースファイルの名前に置き換えます。
    <source_file> を標準入力に置き換えることもできます。次に、**rustfmt** は標準出力に出力を提供します。

注記

デフォルトでは、**rustfmt** は詳細を表示したりバックアップを作成したりせずに、影響を受けるファイルを変更します。詳細を表示してバックアップを作成するには、**--write-mode value** を指定して **rustfmt** を実行します。

3.3. CARGO ビルドツールでの RUSTFMT の使用

Cargo で **rustfmt** ツールを使用して、Rust ソースファイルとそのすべての依存関係をフォーマットします。
別の方法として、**rustfmt** をスタンドアロンツールとして使用します。詳細は、スタンドアロンツールとしての rustfmt の使用 を参照してください。

前提条件

- 既存の Rust プロジェクトがある。
  Rust プロジェクトの作成方法は、Rust プロジェクトの作成 を参照してください。

手順

Cargo コードパッケージのすべてのソースファイルをフォーマットするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'cargo fmt'
```

- Red Hat Enterprise Linux 8 の場合

```
$ cargo fmt
```

- Red Hat Enterprise Linux 9 の場合

```
$ cargo fmt
```

注記

rustfmt フォーマットオプションを変更するには、プロジェクトディレクトリーに設定ファイル **rustfmt.toml** を作成し、設定をファイルに追加します。

3.4. 関連情報

- **rustfmt** のヘルプページを表示するには、以下のコマンドを実行します。
  - Red Hat Enterprise Linux 7 の場合

```
$ scl enable rust-toolset-1.62 'rustfmt --help'
```

  - Red Hat Enterprise Linux 8 の場合

```
$ rustfmt --help
```

  - Red Hat Enterprise Linux 9 の場合

```
$ rustfmt --help
```

- **rustfmt** ツールを設定するには、ファイル **Configurations.md** を編集します。
  - Red Hat Enterprise Linux 7 では、以下のパスの下に見つけることができます。
    /opt/rh/rust-toolset-1.62/root/usr/share/doc/rust-toolset-1.62-rustfmt-1.62.1/Configurations.md
  - Red Hat Enterprise Linux 8 では、以下のパスの下に見つけることができます。
    /usr/share/doc/rustfmt/Configurations.md
  - Red Hat Enterprise Linux 9 では、以下のパスの下に見つけることができます。
    /usr/share/doc/rustfmt/Configurations.md

第4章 RHEL 8 の RUST TOOLSET を使用したコンテナーイメージ

RHEL 8 では、Containerfiles を使用して、独自の Rust Toolset コンテナーイメージを Red Hat Universal Base Images (UBI) コンテナー上にビルドできます。

4.1. RHEL 8 での RUST TOOLSET のコンテナーイメージの作成

RHEL 8 では、Rust Toolset パッケージは、Red Hat Universal Base Images (UBI) リポジトリーの一部です。コンテナーのサイズを小さく保つには、Rust Toolset 全体ではなく、個々のパッケージのみをインストールします。

前提条件

- 既存の Containerfile がある。
  Containerfiles 作成の詳細は、Dockerfile reference ページを参照してください。

手順

- Red Hat Container Catalog にアクセスします。
- UBI を選択します。
- **Get this image** をクリックして、指示に従います。
- Rust Toolset を含むコンテナーを作成するには、以下の行を Containerfile に追加します。

```
FROM registry.access.redhat.com/ubi8/ubi:latest

RUN yum install -y rust-toolset
```

- 個々のパッケージのみを含むコンテナーイメージを作成するには、以下の行を Containerfile に追加します。

```
RUN yum install <package-name>
```

- **<package_name>** をインストールするパッケージの名前に置き換えます。

4.2. 関連情報

- Red Hat UBI イメージの詳細は、コンテナーイメージの使用 を参照してください。
- Red Hat UBI リポジトリーの詳細は、Universal Base Images (UBI): イメージ、リポジトリー、パッケージ、およびソースコード を参照してください。

第5章 RUST 1.62.1 TOOLSET の変更点

Rust Toolset が、RHEL 7 および RHEL 8、RHEL 9 のバージョン 1.58.0 から 1.62.1 に更新されました。

主な変更点は、以下のとおりです。

- 分割代入では、代入の左側にある既存の変数にパターンを代入できます。たとえば、タプル代入は変数にスワップできます: **(a、b) = (b、a);**
- **core::arch::asm** マクロを使用して、インラインアセンブリーが 64 ビット x86 および 64 ビット ARM でサポートされるようになりました。詳細は、The Rust Reference の Inline assembly の章を参照してください。
- 列挙は、明示的にアノテーションが付けられた **#[default]** バリアントを使用して **Default** トレイトを派生できるようになりました。
- **Mutex**、**CondVar**、および **RwLock** は、pthreads ではなくカスタム **futex** ベースの実装を使用するようになり、Rust 言語保証によって新たな最適化が可能になりました。
- Rust は、新しく安定化された **Termination** トレイトを実装するユーザー定義型を含む、**main** からのカスタム終了コードをサポートするようになりました。
- Cargo は、依存関係機能のより詳細な制御をサポートしています。dep: プレフィックスは、それを機能として公開することなく、オプションの依存関係を参照できます。また、"? は、"package-name?/feature-name" のように、その依存関係が他の場所で有効になっている場合にのみ、依存関係の機能を有効にします。
- Cargo には、依存関係を **Cargo.toml** に追加するための新しい **cargo add** サブコマンドがあります。

更新に関する詳細情報は、一連のアップストリームリリース発表を参照してください。

- Announcing Rust 1.59.0
- Announcing Rust 1.60.0
- Announcing Rust 1.61.0
- Announcing Rust 1.62.0
- Announcing Rust 1.62.1