Red Hat Developer Tools 1

Using Rust 1.71.1 Toolset

Installing and using Rust 1.71.1 Toolset

Last Updated: 2023-11-15

Red Hat Developer Tools 1 Using Rust 1.71.1 Toolset

Installing and using Rust 1.71.1 Toolset

Jacob Valdez
jvaldez@redhat.com

Legal Notice

Abstract

Rust Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux (RHEL) operating system. Use this guide for an overview of Rust Toolset, to learn how to invoke and use different versions of Rust tools, and to find resources with more in-depth information.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. RUST TOOLSET

Rust Toolset is a Red Hat offering for developers on Red Hat Enterprise Linux (RHEL). It provides the **rustc** compiler for the Rust programming language, the Rust package manager Cargo, the **rustfmt** formatting tool, and required libraries.

For Red Hat Enterprise Linux 8, Rust Toolset is available as a module. Rust Toolset is available as packages for Red Hat Enterprise Linux 9.

1.1. RUST TOOLSET COMPONENTS

The following components are available as part of Rust Toolset:

Name	Version	Description
rust	1.71.1	The Rust compiler front-end for LLVM.
cargo	1.71.1	A build system and dependency manager for Rust.
rustfmt	1.71.1	A tool for automatic formatting of Rust code.

1.2. RUST TOOLSET COMPATIBILITY

Rust Toolset is available for Red Hat Enterprise Linux 8 and Red Hat Enterprise Linux 9 on the following architectures:

- AMD and Intel 64-bit
- 64-bit ARM
- IBM Power Systems, Little Endian
- 64-bit IBM Z

1.3. INSTALLING RUST TOOLSET

Complete the following steps to install Rust Toolset including all development and debugging tools as well as dependent packages. Note that Rust Toolset has a dependency on LLVM Toolset.

Prerequisites

- All available Red Hat Enterprise Linux updates are installed.

Procedure

On Red Hat Enterprise Linux 8, install the **rust-toolset** module by running:

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

On Red Hat Enterprise Linux 9, install the **rust-toolset** package by running:

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

1.4. INSTALLING RUST DOCUMENTATION

The *The Rust Programming Language* book is available as installable documentation.

Prerequisites

- Rust Toolset is installed.
  For more information, see Installing Rust Toolset.

Procedure

To install the **rust-doc** package, run the following command:

- On Red Hat Enterprise Linux 8:

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

  You can find the *The Rust Programming Language* book under the following path:
  /usr/share/doc/rust/html/index.html.
  You can find the API documentation for all Rust code packages under the following path:
  /usr/share/doc/rust/html/std/index.html.

- On Red Hat Enterprise Linux 9:

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

  You can find the *The Rust Programming Language* book under the following path:
  /usr/share/doc/rust/html/index.html.
  You can find the API documentation for all Rust code packages under the following path:
  /usr/share/doc/rust/html/std/index.html.

1.5. INSTALLING CARGO DOCUMENTATION

The *Cargo, Rust's Package Manager* book is available as installable documentation for Cargo.

NOTE

From Rust Toolset 1.71, the **cargo-doc** package is included in the **rust-doc** package.

Prerequisites

- Rust Toolset is installed.
  For more information, see Installing Rust Toolset.

Procedure

- To install the **cargo-doc** package, run:
  - On Red Hat Enterprise Linux 8:
    -

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

You can find the *Cargo, Rust's Package Manager* book under the following path: **/usr/share/doc/cargo/html/index.html**.

- On Red Hat Enterprise Linux 9:

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

You can find the *Cargo, Rust's Package Manager* book under the following path: **/usr/share/doc/cargo/html/index.html**.

1.6. ADDITIONAL RESOURCES

- For more information on the Rust programming language, see the official Rust documentation.

CHAPTER 2. THE CARGO BUILD TOOL

Cargo is a build tool and front end for the Rust compiler **rustc** as well as a package and dependency manager. It allows Rust projects to declare dependencies with specific version requirements, resolves the full dependency graph, downloads packages, and builds as well as tests your entire project.

Rust Toolset is distributed with Cargo 1.71.1.

2.1. THE CARGO DIRECTORY STRUCTURE AND FILE PLACEMENTS

The Cargo build tool uses set conventions for defining the directory structure and file placement within a Cargo package. Running the **cargo new** command generates the package directory structure and templates for both a manifest and a project file. By default, it also initializes a new Git repository in the package root directory.

For a binary program, Cargo creates a directory ***project_name*** containing a text file named **Cargo.toml** and a subdirectory **src** containing a text file named **main.rs**.

Additional resources

- For more information on the Cargo directory structure, see The Cargo Book — Package Layout .
- For in-depth information about Rust code organization, see The Rust Programming Language — Managing Growing Projects with Packages, Crates, and Modules.

2.2. CREATING A RUST PROJECT

Create a new Rust project that is set up according to the Cargo conventions. For more information on Cargo conventions, see Cargo directory structure and file placements .

Procedure

Create a Rust project by running the following command:

- On Red Hat Enterprise Linux 8:

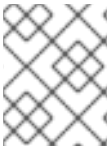
```
$ cargo new --bin <project_name>
```

  - Replace **<*project_name*>** with your project name.

- On Red Hat Enterprise Linux 9:

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

  - Replace **<*project_name*>** with your project name.



NOTE

To edit the project code, edit the main executable file **main.rs** and add new source files to the **src** subdirectory.

Additional resources

- For information on configuring your project and adding dependencies, see Configuring Rust project dependencies.

2.3. CREATING A RUST LIBRARY PROJECT

Complete the following steps to create a Rust library project using the Cargo build tool.

Procedure

To create a Rust library project, run the following command:

- On Red Hat Enterprise Linux 8:

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

  - Replace **<project_name>** with the name of your Rust project.

- On Red Hat Enterprise Linux 9:

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

  - Replace **<project_name>** with the name of your Rust project.

NOTE

To edit the project code, edit the source file, **lib.rs**, in the **src** subdirectory.

Additional resources

- Managing Growing Projects with Packages, Crates, and Modules

2.4. BUILDING A RUST PROJECT

Build your Rust project using the Cargo build tool. Cargo resolves all dependencies of your project, downloads missing dependencies, and compiles it using the **rustc** compiler.

By default, projects are built and compiled in debug mode. For information on compiling your project in release mode, see Building a Rust project in release mode .

Prerequisites

- An existing Rust project.
  For information on how to create a Rust project, see Creating a Rust project.

Procedure

- To build a Rust project managed by Cargo, run in the project directory:
  - On Red Hat Enterprise Linux 8:

```
$ cargo build
```

  - On Red Hat Enterprise Linux 9:

```
$ cargo build
```

- To verify that your Rust program can be built when you do not need to build an executable file, run:

```
$ cargo check
```

2.5. BUILDING A RUST PROJECT IN RELEASE MODE

Build your Rust project in release mode using the Cargo build tool. Release mode is optimizing your source code and can therefore increase compilation time while ensuring that the compiled binary will run faster. Use this mode to produce optimized artifacts suitable for release and production.
Cargo resolves all dependencies of your project, downloads missing dependencies, and compiles it using the **rustc** compiler.

For information on compiling your project in debug mode, see Building a Rust project.

Prerequisites

- An existing Rust project.
  For information on how to create a Rust project, see Creating a Rust project.

Procedure

- To build the project in release mode, run:
  - On Red Hat Enterprise Linux 8:

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

  - On Red Hat Enterprise Linux 9:

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

- To verify that your Rust program can be build when you do not need to build an executable file, run:

```
$ cargo check
```

2.6. RUNNING A RUST PROGRAM

Run your Rust project using the Cargo build tool. Cargo first rebuilds your project and then runs the resulting executable file. If used during development, the **cargo run** command correctly resolves the output path independently of the build mode.

Prerequisites

- A built Rust project.
  For information on how to build a Rust project, see Building a Rust project.

Procedure

To run a Rust program managed as a project by Cargo, run in the project directory:

- On Red Hat Enterprise Linux 8:

```
$ cargo run
```

- On Red Hat Enterprise Linux 9:

```
$ cargo run
```

NOTE

If your program has not been built yet, Cargo builds your program before running it.

2.7. TESTING A RUST PROJECT

Test your Rust program using the Cargo build tool. Cargo first rebuilds your project and then runs the tests found in the project. Note that you can only test functions that are free, monomorphic, and take no arguments. The function return type must be either **()** or **Result<(), E> where E: Error**.

By default, Rust projects are tested in debug mode. For information on testing your project in release mode, see Testing a Rust project in release mode .

Prerequisites

- A built Rust project.
  For information on how to build a Rust project, see Building a Rust project .

Procedure

- Add the test attribute **#[test]** in front of your function.
- To run tests for a Rust project managed by Cargo, run in the project directory:
  - On Red Hat Enterprise Linux 8:

```
$ cargo test
```

  - On Red Hat Enterprise Linux 9:

```
$ cargo test
```

Additional resources

- For more information on performing tests in your Rust project, see The Rust Reference – Testing attributes.

2.8. TESTING A RUST PROJECT IN RELEASE MODE

Test your Rust program in release mode using the Cargo build tool. Release mode is optimizing your source code and can therefore increase compilation time while ensuring that the compiled binary will run faster. Use this mode to produce optimized artifacts suitable for release and production.

Cargo first rebuilds your project and then runs the tests found in the project. Note that you can only test functions that are free, monomorphic, and take no arguments. The function return type must be either **()** or **Result<(), E> where E: Error**.

For information on testing your project in debug mode, see Testing a Rust project.

Prerequisites

- A built Rust project.
  For information on how to build a Rust project, see Building a Rust project.

Procedure

- Add the test attribute **#[test]** in front of your function.
- To run tests for a Rust project managed by Cargo in release mode, run in the project directory:
  - On Red Hat Enterprise Linux 8:

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

  - On Red Hat Enterprise Linux 9:

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

Additional resources

- For more information on performing tests in your Rust project, see The Rust Reference – Testing attributes.

2.9. CONFIGURING RUST PROJECT DEPENDENCIES

Configure the dependencies of your Rust project using the Cargo build tool. To specify dependencies for a project managed by Cargo, edit the file **Cargo.toml** in the project directory and rebuild your project. Cargo downloads the Rust code packages and their dependencies, stores them locally, builds all of the project source code including the dependency code packages, and runs the resulting executable.

Prerequisites

- A built Rust project.
  For information on how to build a Rust project, see Building a Rust project.

Procedure

1. In your project directory, open the file **Cargo.toml**.
2. Move to the section labelled **[dependencies]**.
   Each dependency is listed on a new line in the following format:

   ```
   crate_name = version
   ```

   Rust code packages are called crates.
3. Edit your dependencies.

4. Rebuild your project by running:
   - On Red Hat Enterprise Linux 8:

     ```
     $ cargo build
     ```

   - On Red Hat Enterprise Linux 9:

     ```
     $ cargo build
     ```

5. Run your project by using the following command:
   - On Red Hat Enterprise Linux 8:

     ```
     $ cargo run
     ```

   - On Red Hat Enterprise Linux 9:

     ```
     $ cargo run
     ```

Additional resources

- For more information on configuring Rust dependencies, see The Cargo Book — Specifying Dependencies.

2.10. BUILDING DOCUMENTATION FOR A RUST PROJECT

Use the Cargo tool to generate documentation from comments in your source code that are marked for extraction. Note that documentation comments are extracted only for public functions, variables, and members.

Prerequisites

- A built Rust project.
  For information on how to build a Rust project, see Building a Rust project.
- Configured dependencies.
  For more information on configuring dependencies, see Configuring Rust project dependencies.

Procedure

- To mark comments for extraction, use three slashes /// and place your comment in the beginning of the line it is documenting.
  Cargo supports the Markdown language for your comments.
- To build project documentation using Cargo, run in the project directory:
  - On Red Hat Enterprise Linux 8:

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

  - On Red Hat Enterprise Linux 9:

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

The generated documentation is located in the **.target/doc** directory.

Additional resources

- For more information on building documentation using Cargo, see The Rust Programming Language – Making Useful Documentation Comments.

2.11. COMPILING CODE INTO A WEBASSEMBLY BINARY WITH RUST ON RED HAT ENTERPRISE LINUX 8 AND RED HAT ENTERPRISE LINUX 9 BETA

Complete the following steps to install the WebAssembly standard library.

Prerequisites

- Rust Toolset is installed.
  For more information, see Installing Rust Toolset.

Procedure

- To install the WebAssembly standard library, run:
  - On Red Hat Enterprise Linux 8:

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

  - On Red Hat Enterprise Linux 9:

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

- To use WebAssembly with Cargo, run:
  - On Red Hat Enterprise Linux 8:

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

    Replace <*command*> with the Cargo command you want to run.

  - On Red Hat Enterprise Linux 9:

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

    Replace <*command*> with the Cargo command you want to run.

Additional resources

- For more information on WebAssembly, see the official Rust and WebAssembly documentation or the Rust and WebAssembly book.

2.12. VENDORING RUST PROJECT DEPENDENCIES

Create a local copy of the dependencies of your Rust project for offline redistribution and reuse using the Cargo build tool. This procedure is called vendoring project dependencies. The vendored dependencies including Rust code packages for building your project on a Windows operating system are located in the **vendor** directory. Vendored dependencies can be used by Cargo without any connection to the internet.

Prerequisites

- A built Rust project.
  For information on how to build a Rust project, see Building a Rust project.
- Configured dependencies.
  For more information on configuring dependencies, see Configuring Rust project dependencies.

Procedure

To vendor your Rust project with dependencies using Cargo, run in the project directory:

- On Red Hat Enterprise Linux 8:

```
$ cargo vendor
```

- On Red Hat Enterprise Linux 9:

```
$ cargo vendor
```

2.13. ADDITIONAL RESOURCES

- For more information on Cargo, see the Official Cargo Guide.
- To display the manual page included in Rust Toolset, run:
  - For Red Hat Enterprise Linux 8:

```
$ man cargo
```

  - For Red Hat Enterprise Linux 9:

```
$ man cargo
```

CHAPTER 3. THE RUSTFMT FORMATTING TOOL

With the **rustfmt** formatting tool, you can automatically format the source code of your Rust programs. You can use **rusftmt** either as a standalone tool or with Cargo.

3.1. INSTALLING RUSTFMT

Complete the following steps to install the **rustfmt** formatting tool.

Prerequisites

- Rust Toolset is installed.
  For more information, see Installing Rust Toolset.

Procedure

Run the following command to install **rustfmt**:

- On Red Hat Enterprise Linux 8:

```
# yum install rustfmt
```

- On Red Hat Enterprise Linux 9:

```
# dnf install rustfmt
```

3.2. USING RUSTFMT AS A STANDALONE TOOL

Use **rustfmt** as a standalone tool to format a Rust source file and all its dependencies. As an alternative, use **rustfmt** with the Cargo build tool. For more information, see Using rustfmt with Cargo.

Prerequisites

- An existing Rust project.
  For information on how to create a Rust project, see Creating a Rust project.

Procedure

To format a Rust source file using **rustfmt** as a standalone tool, run the following command:

- On Red Hat Enterprise Linux 8:

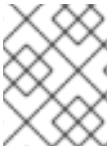
```
$ rustfmt <source-file>
```

  - Replace ***<source_file>*** with the name of your source file.
    Alternatively, you can replace ***<source_file>*** with standard input. **rustfmt** then provides its output in standard output.

- On Red Hat Enterprise Linux 9:

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

- Replace ***<source_file>*** with the name of your source file.
  Alternatively, you can replace ***<source_file>*** with standard input. **rustfmt** then provides its output in standard output.



NOTE

By default, **rustfmt** modifies the affected files without displaying details or creating backups. To display details and create backups, run **rustfmt** with the **--write-mode *value***.

3.3. USING RUSTFMT WITH THE CARGO BUILD TOOL

Use the **rustfmt** tool with Cargo to format a Rust source file and all its dependencies.
As an alternative, use **rustfmt** as a standalone tool. For more information, see Using rustfmt as a standalone tool.

Prerequisites

- An existing Rust project.
  For information on how to create a Rust project, see Creating a Rust project.

Procedure

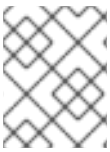
To format all source files in a Cargo code package, run the following command:

- On Red Hat Enterprise Linux 8:

```
$ cargo fmt
```

- On Red Hat Enterprise Linux 9:

```
$ cargo fmt
```



NOTE

To change the **rustfmt** formatting options, create the configuration file **rustfmt.toml** in the project directory and add your configurations to the file.

3.4. ADDITIONAL RESOURCES

- To display the help pages of **rustfmt**, run:
  - On Red Hat Enterprise Linux 8:

```
$ rustfmt --help
```

  - On Red Hat Enterprise Linux 9:

```
$ rustfmt --help
```

- To configure the **rustfmt** tool, create the **rustfmt.toml** configuration file in the project directory and add your configurations to the file. You can find the configuration options in the **Configurations.md** file.
  - On Red Hat Enterprise Linux 8, you can find it under the following path:

/**usr/share/doc/rustfmt/Configurations.md**

- On Red Hat Enterprise Linux 9, you can find it under the following path:
  /**usr/share/doc/rustfmt/Configurations.md**

CHAPTER 4. CONTAINER IMAGES WITH RUST TOOLSET ON RHEL 8

On RHEL 8, you can build your own Rust Toolset container images on top of Red Hat Universal Base Images (UBI) containers using Containerfiles.

4.1. CREATING A CONTAINER IMAGE OF RUST TOOLSET ON RHEL 8

On RHEL 8, Rust Toolset packages are part of the Red Hat Universal Base Images (UBIs) repositories. To keep the container size small, install only individual packages instead of the entire Rust Toolset.

Prerequisites

- An existing Containerfile.
  For more information on creating Containerfiles, see the Dockerfile reference page.

Procedure

- Visit the Red Hat Container Catalog .
- Select a UBI.
- Click **Get this image** and follow the instructions.
- To create a container containing Rust Toolset, add the following lines to your Containerfile:

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

RUN yum install -y rust-toolset
```

- To create a container image containing an individual package only, add the following lines to your Containerfile:

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

- Replace ***<package_name>*** with the name of the package you want to install.

4.2. ADDITIONAL RESOURCES

- For more information on Red Hat UBI images, see Working with Container Images.
- For more information on Red Hat UBI repositories, see Universal Base Images (UBI): Images, repositories, packages, and source code.

CHAPTER 5. CHANGES IN RUST 1.71.1 TOOLSET

Rust Toolset has been updated from version 1.66.1 to 1.71.1 on RHEL 8 and RHEL 9.

Notable changes include:

- A new implementation of multiple producer, single consumer (mpsc) channels to improve performance.
- A new Cargo **sparse** index protocol for more efficient use of the **crates.io** registry.
- New **OnceCell** and **OnceLock** types for one-time value initialization.
- A new **C-unwind** ABI string to enable usage of forced unwinding across Foreign Function Interface (FFI) boundaries.

For detailed information regarding the updates, see the series of upstream release announcements:

- Announcing Rust 1.67.0
- Announcing Rust 1.68.0
- Announcing Rust 1.69.0
- Announcing Rust 1.70.0
- Announcing Rust 1.71.0