The Rust Programming Language

This is the main source code repository for Rust. It contains the compiler, standard library, and documentation.

Note: this README is for *users* rather than *contributors*. If you wish to *contribute* to the compiler, you should read the Getting Started section of the rustc-dev-guide instead. You can ask for help in the #new members Zulip stream.

Quick Start

Read "Installation" from The Book.

Installing from Source

The Rust build system uses a Python script called `x.py` to build the compiler, which manages the bootstrapping process. It lives in the root of the project.

The `x.py` command can be run directly on most systems in the following format:

```
./x.py <subcommand> [flags]
```

This is how the documentation and examples assume you are running `x.py`.

Systems such as Ubuntu 20.04 LTS do not create the necessary `python` command by default when Python is installed that allows `x.py` to be run directly. In that case you can either create a symlink for `python` (Ubuntu provides the `python-is-python3` package for this), or run `x.py` using Python itself:

```
# Python 3
python3 x.py <subcommand> [flags]

# Python 2.7
python2.7 x.py <subcommand> [flags]
```

More information about `x.py` can be found by running it with the `--help` flag or reading the rustc dev guide.

Building on a Unix-like system

1. Make sure you have installed the dependencies:
    - `g++` 5.1 or later or `clang++` 3.5 or later
    - `python` 3 or 2.7
    - GNU `make` 3.81 or later
    - `cmake` 3.13.4 or later
    - `ninja`
    - `curl`
    - `git`
    - `ssl` which comes in `libssl-dev` or `openssl-devel`
    - `pkg-config` if you are compiling on Linux and targeting Linux
2. Clone the source with `git`:

```
git clone https://github.com/rust-lang/rust.git
cd rust
```

3. Configure the build settings:

   The Rust build system uses a file named `config.toml` in the root of the source tree to determine various configuration settings for the build. Copy the default `config.toml.example` to `config.toml` to get started.

   ```
   cp config.toml.example config.toml
   ```

   If you plan to use `x.py install` to create an installation, it is recommended that you set the `prefix` value in the `[install]` section to a directory.

   Create install directory if you are not installing in default directory

4. Build and install:

   ```
   ./x.py build && ./x.py install
   ```

   When complete, `./x.py install` will place several programs into `$PREFIX/bin`: `rustc`, the Rust compiler, and `rustdoc`, the API-documentation tool. This install does not include Cargo, Rust's package manager. To build and install Cargo, you may run `./x.py install cargo` or set the `build.extended` key in `config.toml` to `true` to build and install all tools.

Building on Windows

There are two prominent ABIs in use on Windows: the native (MSVC) ABI used by Visual Studio, and the GNU ABI used by the GCC toolchain. Which version of Rust you need depends largely on what C/C++ libraries you want to interoperate with: for interop with software produced by Visual Studio use the MSVC build of Rust; for interop with GNU software built using the MinGW/MSYS2 toolchain use the GNU build.

MinGW

MSYS2 can be used to easily build Rust on Windows:

1. Grab the latest MSYS2 installer and go through the installer.

2. Run `mingw32_shell.bat` or `mingw64_shell.bat` from wherever you installed MSYS2 (i.e. `C:\msys64`), depending on whether you want 32-bit or 64-bit Rust. (As of the latest version of MSYS2 you have to run `msys2_shell.cmd -mingw32` or `msys2_shell.cmd -mingw64` from the command line instead)

3. From this terminal, install the required tools:

   ```
   # Update package mirrors (may be needed if you have a fresh install of MSYS2)
   pacman -Sy pacman-mirrors

   # Install build tools needed for Rust. If you're building a 32-bit compiler,
   # then replace "x86_64" below with "i686". If you've already got git, python,
   ```

```
# or CMake installed and in PATH you can remove them from this list. Note
# that it is important that you do **not** use the 'python2', 'cmake' and
'ninja'
# packages from the 'msys2' subsystem. The build has historically been known
# to fail with these packages.
pacman -S git \
            make \
            diffutils \
            tar \
            mingw-w64-x86_64-python \
            mingw-w64-x86_64-cmake \
            mingw-w64-x86_64-gcc \
            mingw-w64-x86_64-ninja
```

4. Navigate to Rust's source code (or clone it), then build it:

```
./x.py build && ./x.py install
```

MSVC

MSVC builds of Rust additionally require an installation of Visual Studio 2017 (or later) so `rustc` can use its linker. The simplest way is to get the Visual Studio, check the "C++ build tools" and "Windows 10 SDK" workload.

(If you're installing cmake yourself, be careful that "C++ CMake tools for Windows" doesn't get included under "Individual components".)

With these dependencies installed, you can build the compiler in a `cmd.exe` shell with:

```
python x.py build
```

Currently, building Rust only works with some known versions of Visual Studio. If you have a more recent version installed and the build system doesn't understand, you may need to force rustbuild to use an older version. This can be done by manually calling the appropriate vcvars file before running the bootstrap.

```
CALL "C:\Program Files (x86)\Microsoft Visual
Studio\2019\Community\VC\Auxiliary\Build\vcvars64.bat"
python x.py build
```

Specifying an ABI

Each specific ABI can also be used from either environment (for example, using the GNU ABI in PowerShell) by using an explicit build triple. The available Windows build triples are:

- GNU ABI (using GCC)
  - `i686-pc-windows-gnu`
  - `x86_64-pc-windows-gnu`
- The MSVC ABI
  - `i686-pc-windows-msvc`
  - `x86_64-pc-windows-msvc`

The build triple can be specified by either specifying `--build=<triple>` when invoking `x.py` commands, or by copying the `config.toml` file (as described in Installing From Source), and modifying the `build` option under the `[build]` section.

Configure and Make

While it's not the recommended build system, this project also provides a configure script and makefile (the latter of which just invokes `x.py`).

```
./configure
make && sudo make install
```

When using the configure script, the generated `config.mk` file may override the `config.toml` file. To go back to the `config.toml` file, delete the generated `config.mk` file.

Building Documentation

If you'd like to build the documentation, it's almost the same:

```
./x.py doc
```

The generated documentation will appear under `doc` in the `build` directory for the ABI used. I.e., if the ABI was `x86_64-pc-windows-msvc`, the directory will be `build\x86_64-pc-windows-msvc\doc`.

Notes

Since the Rust compiler is written in Rust, it must be built by a precompiled "snapshot" version of itself (made in an earlier stage of development). As such, source builds require a connection to the Internet, to fetch snapshots, and an OS that can execute the available snapshot binaries.

Snapshot binaries are currently built and tested on several platforms:

Platform / Architecture	x86	x86_64
Windows (7, 8, 10, ...)	✓	✓
Linux (kernel 2.6.32, glibc 2.11 or later)	✓	✓
macOS (10.7 Lion or later)	(*)	✓

(*): Apple dropped support for running 32-bit binaries starting from macOS 10.15 and iOS 11. Due to this decision from Apple, the targets are no longer useful to our users. Please read our blog post for more info.

You may find that other platforms work, but these are our officially supported build environments that are most likely to work.

Getting Help

The Rust community congregates in a few places:

- Stack Overflow - Direct questions about using the language.
- users.rust-lang.org - General discussion and broader questions.

- /r/rust - News and general discussion.

Contributing

If you are interested in contributing to the Rust project, please take a look at the Getting Started guide in the rustc-dev-guide.

License

Rust is primarily distributed under the terms of both the MIT license and the Apache License (Version 2.0), with portions covered by various BSD-like licenses.

See LICENSE-APACHE, LICENSE-MIT, and COPYRIGHT for details.

Trademark

The Rust Foundation owns and protects the Rust and Cargo trademarks and logos (the “Rust Trademarks”).

If you want to use these names or brands, please read the media guide.

Third-party logos may be subject to third-party copyrights and trademarks. See Licenses for details.