E

Editions

In Chapter 1, you saw that cargo new adds a bit of metadata to your Cargo.toml file about an edition. This appendix talks about what that means!

The Rust language and compiler have a six-week release cycle, meaning users get a constant stream of new features. Other programming languages release larger changes less often; Rust releases smaller updates more frequently. After a while, all of these tiny changes add up. But from release to release, it can be difficult to look back and say, “Wow, between Rust 1.10 and Rust 1.31, Rust has changed a lot!”

Every two or three years, the Rust team produces a new Rust edition. Each edition brings together the features that have landed into a clear package with fully updated documentation and tooling. New editions ship as part of the usual six-week release process.

Editions serve different purposes for different people:

* For active Rust users, a new edition brings together incremental changes into an easy-to-understand package.
* For non-users, a new edition signals that some major advancements have landed, which might make Rust worth another look.
* For those developing Rust, a new edition provides a rallying point for the project as a whole.

At the time of this writing, three Rust editions are available: Rust 2015, Rust 2018, and Rust 2021. This book is written using Rust 2021 edition idioms.

The edition key in Cargo.toml indicates which edition the compiler should use for your code. If the key doesn’t exist, Rust uses 2015 as the edition value for backward compatibility reasons.

Each project can opt in to an edition other than the default 2015 edition. Editions can contain incompatible changes, such as including a new keyword that conflicts with identifiers in code. However, unless you opt in to those changes, your code will continue to compile even as you upgrade the Rust compiler version you use.

All Rust compiler versions support any edition that existed prior to that compiler’s release, and they can link crates of any supported editions together. Edition changes only affect the way the compiler initially parses code. Therefore, if you’re using Rust 2015 and one of your dependencies uses Rust 2018, your project will compile and be able to use that dependency. The opposite situation, where your project uses Rust 2018 and a dependency uses Rust 2015, works as well.

To be clear: most features will be available on all editions. Developers using any Rust edition will continue to see improvements as new stable releases are made. However, in some cases, mainly when new keywords are added, some new features might only be available in later editions. You will need to switch editions if you want to take advantage of such features.

For more details, The Edition Guide at [https://doc.rust-lang.org/stable/edition-guide](https://doc.rust-lang.org/stable/edition-guide/) is a complete book about editions that enumerates the differences between editions and explains how to automatically upgrade your code to a new edition via cargo fix.