The Rust Programming Language Blog

Announcing Rust 1.19

Jul 20, 2017 • The Rust Core Team

The Rust team is happy to announce the latest version of Rust, 1.19.0. Rust is a systems programming language focused on safety, speed, and concurrency.

If you have a previous version of Rust installed, getting Rust 1.19 is as easy as:

```
$ rustup update stable
```

If you don't have it already, you can get `rustup` from the appropriate page on our website, and check out the detailed release notes for 1.19.0 on GitHub.

What's in 1.19.0 stable

Rust 1.19.0 has some long-awaited features, but first, a note for our Windows users. On Windows, Rust relies on `link.exe` for linking, which you can get via the "Microsoft Visual C++ Build Tools." With the recent release of Visual Studio 2017, the directory structure for these tools has changed. As such, to use Rust, you had to stick with the 2015 tools or use a workaround (such as running `vcvars.bat`). In 1.19.0, `rustc` now knows how to find the 2017 tools, and so they work without a workaround.

On to new features! Rust 1.19.0 is the first release that supports `union`s:

```
union MyUnion {
    f1: u32,
    f2: f32,
}
```

Unions are kind of like `enum`s, but they are "untagged". Enums have a "tag" that stores which variant is the correct one at runtime; unions elide this tag.

Since we can interpret the data held in the union using the wrong variant and Rust can't check this for us, that means reading or writing a union's field is unsafe:

```
let u = MyUnion { f1: 1 };

unsafe { u.f1 = 5 };

let value = unsafe { u.f1 };
```

Pattern matching works too:

```
fn f(u: MyUnion) {
    unsafe {
        match u {
            MyUnion { f1: 10 } => { println!("ten"); }
            MyUnion { f2 } => { println!("{}", f2); }
        }
    }
}
```

When are unions useful? One major use-case is interoperability with C. C APIs can (and depending on the area, often do) expose unions, and so this makes writing API wrappers for those libraries significantly easier. Additionally, from its RFC:

> *A native union mechanism would also simplify Rust implementations of space-efficient or cache-efficient structures relying on value representation, such as machine-word-sized unions using the least-significant bits of aligned pointers to distinguish cases.*

This feature has been long awaited, and there's still more improvements to come. For now, `union`s can only include `Copy` types and may not implement `Drop`. We expect to lift these restrictions in the future.

> *As a side note, have you ever wondered how new features get added to Rust? This feature was suggested by Josh Triplett, and he gave a talk at RustConf 2016 about the process of getting `union`s into Rust. You should check it out!*

In other news, `loop`s can now `break` with a value:

```
// old code
let x;

loop {
    x = 7;
    break;
}

// new code
let x = loop { break 7; };
```

Rust has traditionally positioned itself as an “expression oriented language”, that is, most things are expressions that evaluate to a value, rather than statements. `loop` stuck out as strange in this way, as it was previously a statement.

What about other forms of loops? It’s not yet clear. See its RFC for some discussion around the open questions here.

A smaller feature, closures that do not capture an environment can now be coerced to a function pointer:

```
let f: fn(i32) -> i32 = |x| x + 1;
```

We now produce xz compressed tarballs and prefer them by default, making the data transfer smaller and faster. `gzip`‘d tarballs are still produced in case you can’t use `xz` for some reason.

The compiler can now bootstrap on Android. We’ve long supported Android in various ways, and this continues to improve our support.

Finally, a compatibility note. Way back when we were running up to Rust 1.0, we did a huge push to verify everything that was being marked as stable and as unstable. We overlooked one thing, however: `-Z` flags. The `-Z` flag to the compiler enables unstable flags. Unlike the rest of our stability story, you could still use `-Z` on stable Rust. Back in April of 2016, in Rust 1.8, we made the use of `-Z` on stable or beta produce a warning. Over a year later, we’re fixing this hole in our stability story by disallowing `-Z` on stable and beta.

See the detailed release notes for more.

Library stabilizations

The largest new library feature is the `eprint!` and `eprintln!` macros. These work exactly the same as `print!` and `println!` but instead write to standard error, as opposed to standard output.

Other new features:

- `String` now implements `FromIterator<Cow<'a, str>>` and `Extend<Cow<'a, str>>`
- `Vec` now implements `From<&mut [T]>`
- `Box<[u8]>` now implements `From<Box<str>>`
- `SplitWhitespace` now implements `Clone`
- `[u8]::reverse` is now 5x faster and `[u16]::reverse` is now 1.5x faster

And some freshly-stabilized APIs:

- `OsString::shrink_to_fit`
- `cmp::Reverse`
- `Command::envs`
- `thread::ThreadId`

See the detailed release notes for more.

Cargo features

Cargo mostly received small but valuable improvements in this release. The largest is possibly that Cargo no longer checks out a local working directory for the crates.io index. This should provide smaller file size for the registry and improve cloning times, especially on Windows machines.

Other improvements:

- Build scripts can now add environment variables to the environment the crate is being compiled in. Example: `println!("cargo:rustc-env=FOO=bar");`
- Workspace members can now accept glob file patterns
- Added `--all` flag to the `cargo bench` subcommand to run benchmarks of all the members in a given workspace.
- Added an `--exclude` option for excluding certain packages when using the `--all` option
- The `--features` option now accepts multiple comma or space delimited values.
- Added support for custom target specific runners

See the detailed release notes for more.

Contributors to 1.19.0

Many people came together to create Rust 1.19. We couldn’t have done it without all of you. Thanks!

More about Rust — Jump straight in

The Rust Programming Language Blog

rust-lang
rustlang

Words from the Rust team