

Rust 2018

an epoch release

by Steve Klabnik

What is Rust?



Systems (?)

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

What is a "systems" language anyway?

Empowerment

It wasn't always so clear, but the Rust programming language is fundamentally about empowerment: no matter what kind of code you are writing now, Rust empowers you to reach farther, to program with confidence in a wider variety of domains than you did before.

Take, for example, "systems-level" work that deals with low-level details of memory management, data representation, and concurrency. Traditionally, this realm of programming is seen as arcane, accessible only to a select few who have devoted the necessary years learning to avoid its infamous pitfalls. And even those who practice it do so with caution, lest their code be open to exploits, crashes, or corruption.

Rust breaks down these barriers by eliminating the old pitfalls and providing a friendly, polished set of tools to help you along the way.

Three Audiences

- C and C++ people
- Ruby, Python, and JavaScript people
- Functional people

How Rust is built

Let's talk about how we make Rust!

Branches

Three release channels:

- Stable
- Beta
- Nightly

Stuff lands in nightly, then ends up in beta, then stable.

Branch management

Everything lands on the master branch.

Every night, there's a nightly release.

Every six weeks, beta branches off of master.

Every six weeks, stable branches off of beta.

Robots



bors
bors

Follow

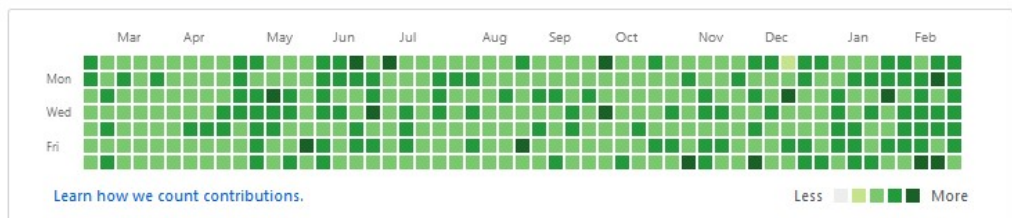
Block or report user

Overview Repositories 0 Stars 0 Followers 87 Following 0

Popular repositories

bors doesn't have any public repositories yet.

40,080 contributions in the last year





Rust highfive robot

rust-highfive

Unfollow

Block or report user

Governance

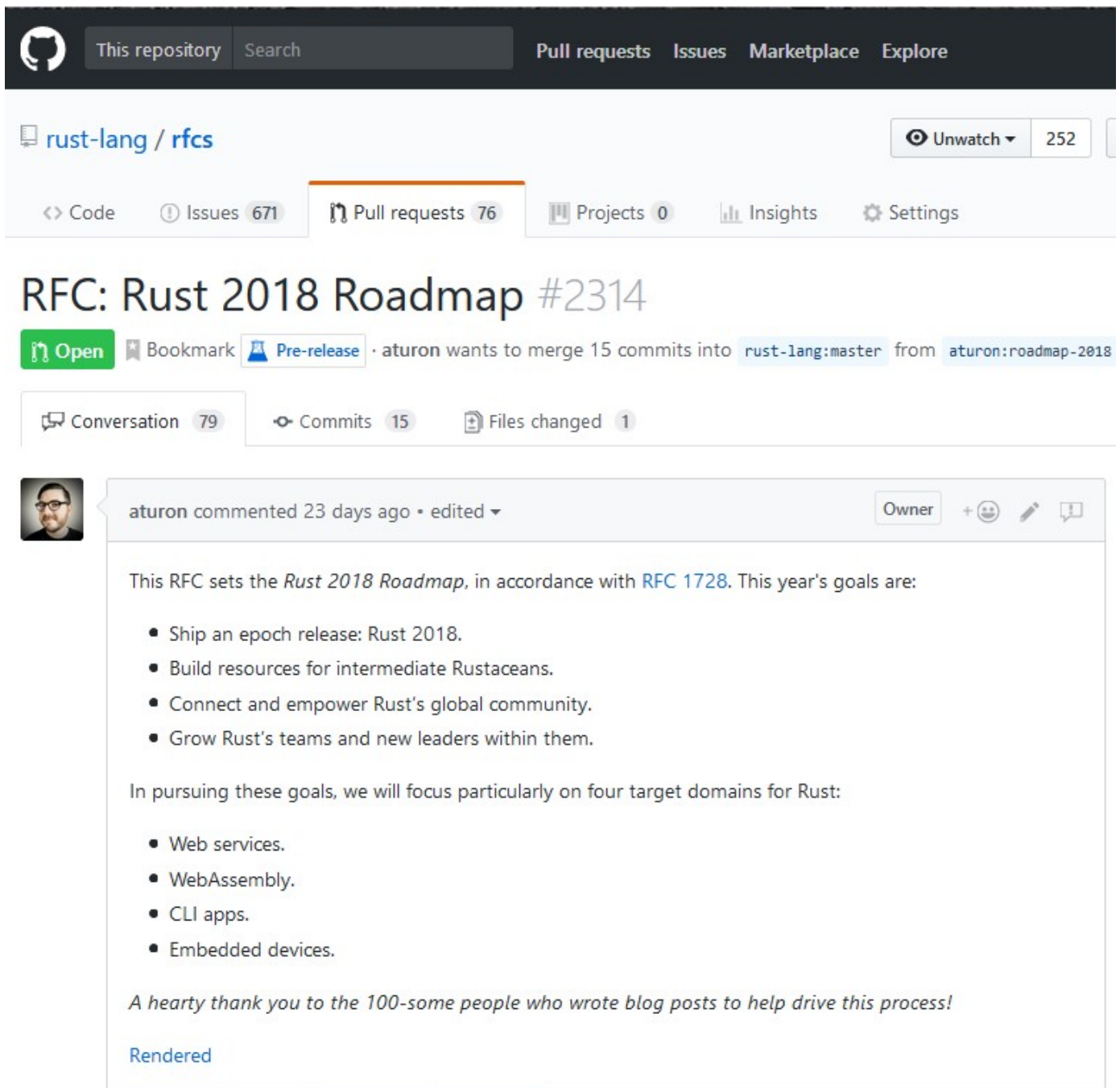
Rust is governed by various teams.

Teams use RFCs to coordinate changes to the language.

Even the core team needs to write RFCs.

Moderation team enforces the Code of Conduct. No core team members can be on the moderation team.

2018 Roadmap



The screenshot shows a GitHub pull request page for the repository 'rust-lang / rfcs'. The pull request is titled 'RFC: Rust 2018 Roadmap #2314' and is in a 'Pre-release' state. It was created by 'aturon' and wants to merge 15 commits into the 'rust-lang:master' branch from the 'aturon:roadmap-2018' branch. The pull request is currently open and has 79 conversations, 15 commits, and 1 file changed. The main content of the pull request is a comment from 'aturon' (commented 23 days ago) that outlines the 'Rust 2018 Roadmap' in accordance with RFC 1728. The comment lists four goals for the year: ship an epoch release, build resources for intermediate Rustaceans, connect and empower the global community, and grow Rust's teams and new leaders. It also lists four target domains for Rust: web services, WebAssembly, CLI apps, and embedded devices. The comment concludes with a hearty thank you to the 100-some people who wrote blog posts to help drive the process.

This repository Search Pull requests Issues Marketplace Explore

rust-lang / rfcs Unwatch 252

Code Issues 671 Pull requests 76 Projects 0 Insights Settings

RFC: Rust 2018 Roadmap #2314

Open Bookmark Pre-release · aturon wants to merge 15 commits into rust-lang:master from aturon:roadmap-2018

Conversation 79 Commits 15 Files changed 1

aturon commented 23 days ago · edited

Owner + 😊 ✎ 🗨

This RFC sets the *Rust 2018 Roadmap*, in accordance with RFC 1728. This year's goals are:

- Ship an epoch release: Rust 2018.
- Build resources for intermediate Rustaceans.
- Connect and empower Rust's global community.
- Grow Rust's teams and new leaders within them.

In pursuing these goals, we will focus particularly on four target domains for Rust:

- Web services.
- WebAssembly.
- CLI apps.
- Embedded devices.

A hearty thank you to the 100-some people who wrote blog posts to help drive this process!

Rendered

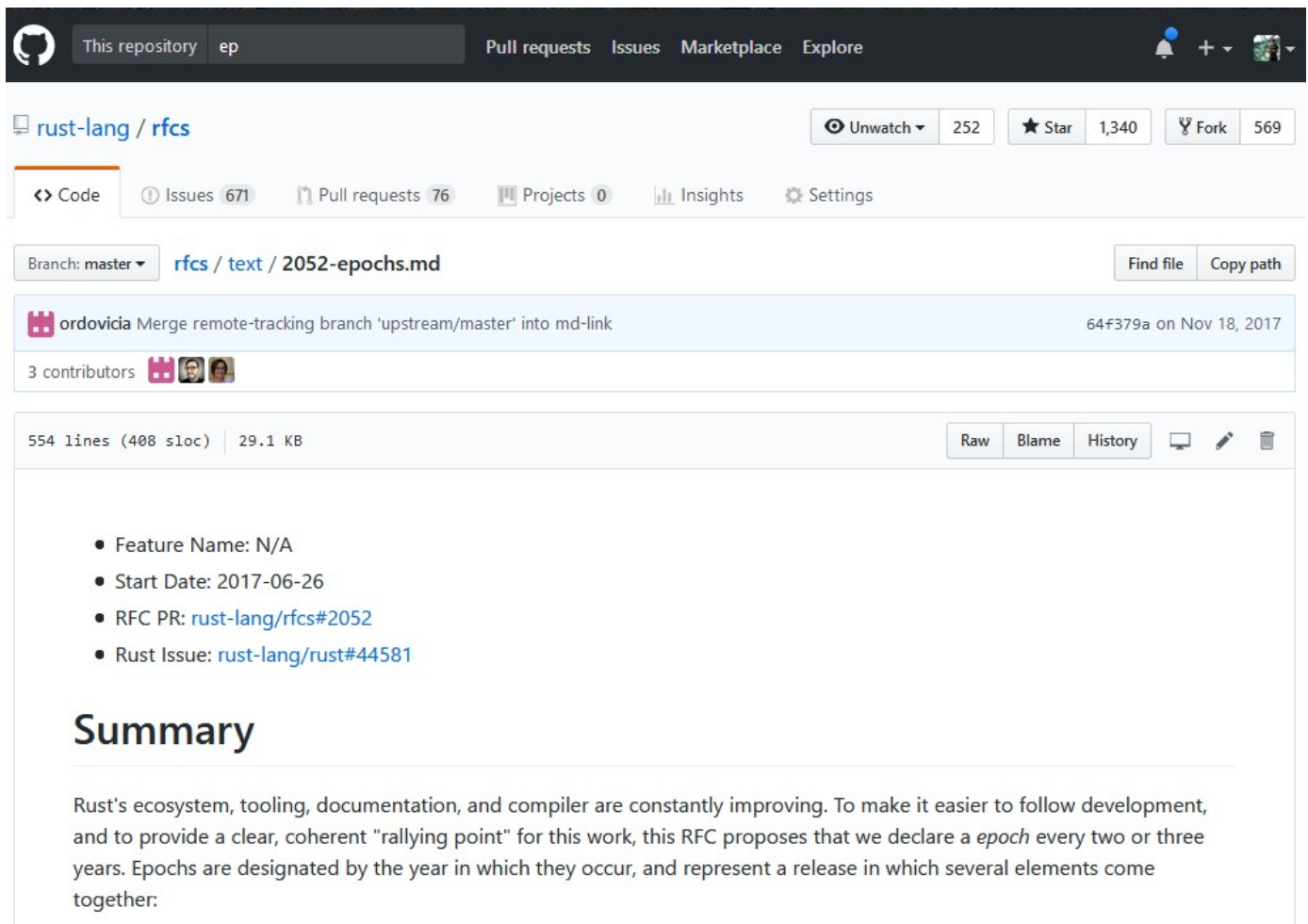
Ship an epoch release: Rust 2018

The biggest item on the roadmap is: Ship Rust 2018

What's an epoch?

Epochs are defined in RFC 2052.

They are fundamentally *rallying points* for Rust development.



The screenshot shows the GitHub interface for the `rust-lang/rfcs` repository. At the top, there are navigation links for Pull requests, Issues, Marketplace, and Explore. The repository name `rust-lang/rfcs` is displayed, along with statistics: 252 Unwatch, 1,340 Stars, and 569 Forks. Below this, there are tabs for Code, Issues (671), Pull requests (76), Projects (0), Insights, and Settings. The current branch is `master`, and the file path is `rfcs/text/2052-epochs.md`. A commit by `ordovicia` is shown, titled "Merge remote-tracking branch 'upstream/master' into md-link", dated Nov 18, 2017. The file size is 29.1 KB and it contains 554 lines. The commit message includes a list of links: Feature Name: N/A, Start Date: 2017-06-26, RFC PR: [rust-lang/rfcs#2052](#), and Rust Issue: [rust-lang/rust#44581](#). A "Summary" section follows, containing a paragraph of text.

- Feature Name: N/A
- Start Date: 2017-06-26
- RFC PR: [rust-lang/rfcs#2052](#)
- Rust Issue: [rust-lang/rust#44581](#)

Summary

Rust's ecosystem, tooling, documentation, and compiler are constantly improving. To make it easier to follow development, and to provide a clear, coherent "rallying point" for this work, this RFC proposes that we declare a *epoch* every two or three years. Epochs are designated by the year in which they occur, and represent a release in which several elements come together:

What's Rust 2018 look like?

The theme of the release is *productivity*.

It will be released in the final third of the year.

Polish what exists

We're not accepting new major features until 2019. 2018's focus will be on polishing features we've accepted but not yet made stable:

- `impl Trait`
- Macros 2.0
- SIMD
- Generators
- Non-Lexical Lifetimes
- Module system revamp

Improved tooling

RLS & Rustfmt 1.0 will hit 1.0.

New website

We're re-doing the website.

Compatibility

Rust cares a *lot* about stability. Epochs keep up this tradition.

Never Rust 2.0

Rust will never have a "2.0" release.

Compatibility is *hard*

We are following in the footsteps of other ecosystems where backwards compatibility is considered sacrosanct: C++ and Java.

Did you know they actually do break things every release?

Not all breaking changes are equal.

Rust 2015 and 2018 inteop perfectly

Compiler supports epochs via a flag.

Cargo will put the epoch into your Cargo.toml and pass it along.

Your code can use Rust 2018, and you can use 2015 libraries.

Your code can be set to 2015, and use 2018 libraries.

Transitioning

If your code compiles without warnings on Rust 2015, it will compile in Rust 2018.

Build resources for intermediate Rustaceans

We will write documentation and build examples that help programmers go from basic knowledge of Rust's mechanics to knowing how to wield it effectively.

We, as a community, should work on creating the next level of learning resources to help folks deploy Rust to production with confidence. (@integer32)

This includes discussions on how to structure big projects in Rust and Rust-specific design patterns. I want to read more about professional Rust usage and see case-studies from various industries. (@mre)

Once you have a grasp of what knobs do what in the language, how do you learn what's considered "proper", or what structures people have found to make future maintenance easier? (@QuietMisdreavus)

Connect and empower Rust's global community

Rust is more than just the Rust team!

"Additionally, more venues should be created to work with production users to gather regular feedback in a convenient, scalable way." (Integer 32)

We should ask how to improve support for local meetups to strengthen community cohesion. (@llogiq)

Mentorship does a lot to help underrepresented groups of people. (@blackdijkstra)

“What can I, or other people not one the core team, do to help stabilize Rust?”
(willmurphyscode)

Rust is very new and most of the documentation coming out is probably going to be in English because it is the most widely used language. However, we have to acknowledge that we don't only have english speakers in the rust community and it would be great if some of the text was translated to accommodate non English speakers. (@blackdijkstra)

Grow Rust's Teams and new leaders within them

There's a lot of work to do! To get it done, we'll need help, and from more people than what we currently have.

Revise the teams

[This just happend.](#)

in other words, we're developing middle management (@quietmisdreavus)

No more 'subteams', just 'teams'.

Teams can have teams.

Core and Moderation at the root, other teams below.

- Docs
 - TRPL
 - Reference
 - std
 - RBE
 - nomicon

"working groups" for focused tasks.

Better resources and mentoring

New book on compiler contribution.

Systematizing mentorship.

Four areas of focus

To get stuff done, you gotta *focus*.

We'll be spinning up dedicated working groups, reporting to the core team, to work on these areas.

That doesn't mean other stuff won't get done, it just won't be a priority for the project.

Web Services

Repeat from last year!

We made huge steps, but there's more to do. It's worth continuing this year, because it's a heavy area for production users.

Last year was about building foundations, this year is about the end-to-end experience.

Async/await



```
#[async]
fn fetch_rust_lang(client: hyper::Client) -> io::Result<String> {
    let response = await!(client.get("https://www.rust-lang.org"))?;

    if !response.status().is_success() {
        return Err(io::Error::new(io::ErrorKind::Other, "request failed"))
    }

    let body = await!(response.body().concat())?;

    let string = String::from_utf8(body)?;

    Ok(string)
}
```

WebAssembly

WebAssembly lets you run Rust in the browser.

We introduced `wasm32-unknown-unknown` late last year. Let's make it awesome!

This year, writing Rust for the web should be a pleasant experience with great tooling.

Systems languages are tied to platforms

Systems languages give you deep access to a particular platform. C rose as UNIX rose.

WebAssembly is a new, rising platform. It has no "default" systems language.

Rust has a number of competitive advantages in this space.

Closely related to embedded

It's also worth pushing on wasm because wasm is effectively an embedded platform.

CLI Apps

Rust is already a great language to write CLI apps:

- Portability
- Reliability
- Static binaries

Quicli

We are going to create a small CLI tool that outputs the first n lines of a given file.



```
#[macro_use]
extern crate quicli;

use quicli::prelude::*;

#[derive(Debug, StructOpt)]
struct Cli {
    // Add a CLI argument `--count`/`-n` that defaults to 3, and has this help text:
    /// How many lines to get
    #[structopt(long = "count", short = "n", default_value = "3")]
    count: usize,
    // Add a positional argument that the user has to supply:
    /// The file to read
    file: String,
    /// Pass many times for more log output
    #[structopt(long = "verbose", short = "v", parse(from_occurrences))]
    verbosity: u8,
}

main!(|args: Cli, log_level: verbosity| {
    read_file(&args.file)?
        .lines()
        .take(args.count)
        .for_each(|line| println!("{}", line));
});
```

Embedded devices

Rust is pretty great on embedded, but requires nightly.

We can't make *everything* stable for embedded this year, but a basic story should be possible.

- "unfork xargo"
- `panic_fmt`
- Better support for `no_std` across the ecosystem

Thanks!

There's a lot to look forward to this year!

Thanks for having me ♥