

Rust

**The most exciting new programming language for years
(what, really?)**

**Ian Jackson GNU/Citrix/Xen/Debian GHM Madrid
September 2019**

Who am I ?

curmudgeon and pessimist

Perl, C, Tcl, Python, bash

some C++, Haskell, Ocaml, asm, JS, Lisp, Java...

**Manual
memory
management**

Unsafe

C

C++

Assembler

**new
malloc**

**free
delete**

GC

Safe

Python JS

Perl Java

Ocaml Lisp

Haskell

**new
Class()
implicit allocation**

**freed after last
referent goes away**

**Manual
memory
management**

Unsafe

C

C++

Assembler

**new
malloc**

**free
delete**

**Ownership
(borrow
checker)**

Safe

Rust

**new
(etc)**

lifetime
sufficiency
checked

GC

Safe

Python JS

Perl Java

Ocaml Lisp

Haskell

**new
Class()
implicit allocation**

freed after last
referent goes away

```
fn main() {  
    let mut s = String::from("hello");  
    change(&mut s);  
    println!("{}", s);  
}
```

```
fn change(some_string: &mut String) {  
    some_string.push_str(", world");  
}
```

```
fn main() {  
    let reference_to_nothing = dangle();  
}  
  
fn dangle() -> &String {  
    let s = String::from("hello");  
    &s  
}
```

error[E0106]: missing lifetime specifier

--> main.rs:5:16

```
5 | fn dangle() -> &String {  
    ^ expected lifetime parameter
```

= help: this function's return type contains a borrowed value, but there is no value for it to be borrowed from
= help: consider giving it a 'static lifetime

```
fn main() {
    let s = String::from("hello");
    change(&s);
    println!("{}", s);
}

fn change(some_string: &String) {
    some_string.push_str(", world");
}
```

```
error[E0596]: cannot borrow immutable borrowed content
             `*some_string` as mutable
```

```
--> error.rs:8:5
```

```
7 | fn change(some_string: &String) {
   |                               ----- use '&mut String' here
   |                               to make mutable
8 |     some_string.push_str(", world");
   |     ^^^^^^^^^^^^^^^ cannot borrow as mutable
```

Other properties of Rust – illustrated

Syntax

Type system

Safety

FFI

inference

unsafe

talking

polymorphism

escape

to C etc.

("generics")

hatch

```
struct Point<T> {  
    x: T,  
    y: T,  
}
```

```
fn main( ) {  
    let i = Point { x: 5, y: 10 };  
    let f = Point { x: 1.0, y: 4.0 };  
}
```

Other properties of Rust – illustrated

Syntax

Type system

inference

polymorphism

("generics")

Safety

unsafe

escape

hatch

FFI

talking

to C etc.

```
struct Point<T> { — definition of a struct type
```

```
    x: T, > definition of the members and their types
```

```
    y: T,
```

```
}
```

construction of struct values

by specifying values of the members

```
fn main( ) {  
    let i = Point { x: 5, y: 10 };  
    let f = Point { x: 1.0, y: 4.0 };  
}
```


Other properties of Rust – illustrated

Syntax

Type system

Safety

FFI

inference

**polymorphism
("generics")**

**unsafe
escape
hatch**

**talking
to C etc.**

```
struct Point<T> {  
    x: T,  
    y: T,  
}  
  
fn main() {  
    let i = Point { x: 5, y: 10 };  
    let f = Point { x: 1.0, y: 4.0 };  
}
```

*struct type is polymorphic
there's Point<T> for any T*

*members are of type T
whatever T is*

*types of i and f not specified by programmer
compiler infers, eg, Point<f64>*

Other properties of Rust – illustrated

Syntax

Type system
inference
polymorphism
("generics")

Safety
unsafe
escape
hatch

FFI
talking
to C etc.

```
struct InsnBytecode {  
    // unsafety: programmer documenting  
    //   the instruction executor assumes that: unsafe code's assumptions  
    //   c is a valid regnum  
    ...  
    c : BytecodeValue,  
    ...  
    unsafe { required to surround any use of unsafe  
        *regs.offset(c as isize) = r; language features or library functions  
    } raw pointer offset calculation  
}
```

bypasses array bounds check

Other properties of Rust – illustrated

FFI
talking
to C etc.

```
#[link(name="glue")]
extern "C" {
    pub fn cxx_chrobak_payne(
        nvertices : size_t,
        edge_data : *const [size_t; 2],
        n_edge_data : size_t,
        ...
    }
    ...
    let ok = unsafe {
        cxx_chrobak_payne(nvertices,
            edges.as_ptr(),
            edges.len(),
            ...
        )
    }
}
```

raw pointer type

calls out of Rust are not safe

types will be checked

Rust

```
extern "C" {
    int cxx_chrobak_payne(const size_t nvertices,
                        const size_t edge_data[][2],
                        const size_t n_edge_data,
                        ...
    )
}
```

C++

Community attitude – Programmer mistakes



**C
compiler/
standards
community**

**Victim
blaming**



**Rust
community**

**Compiler
should help**

July 22, 2017

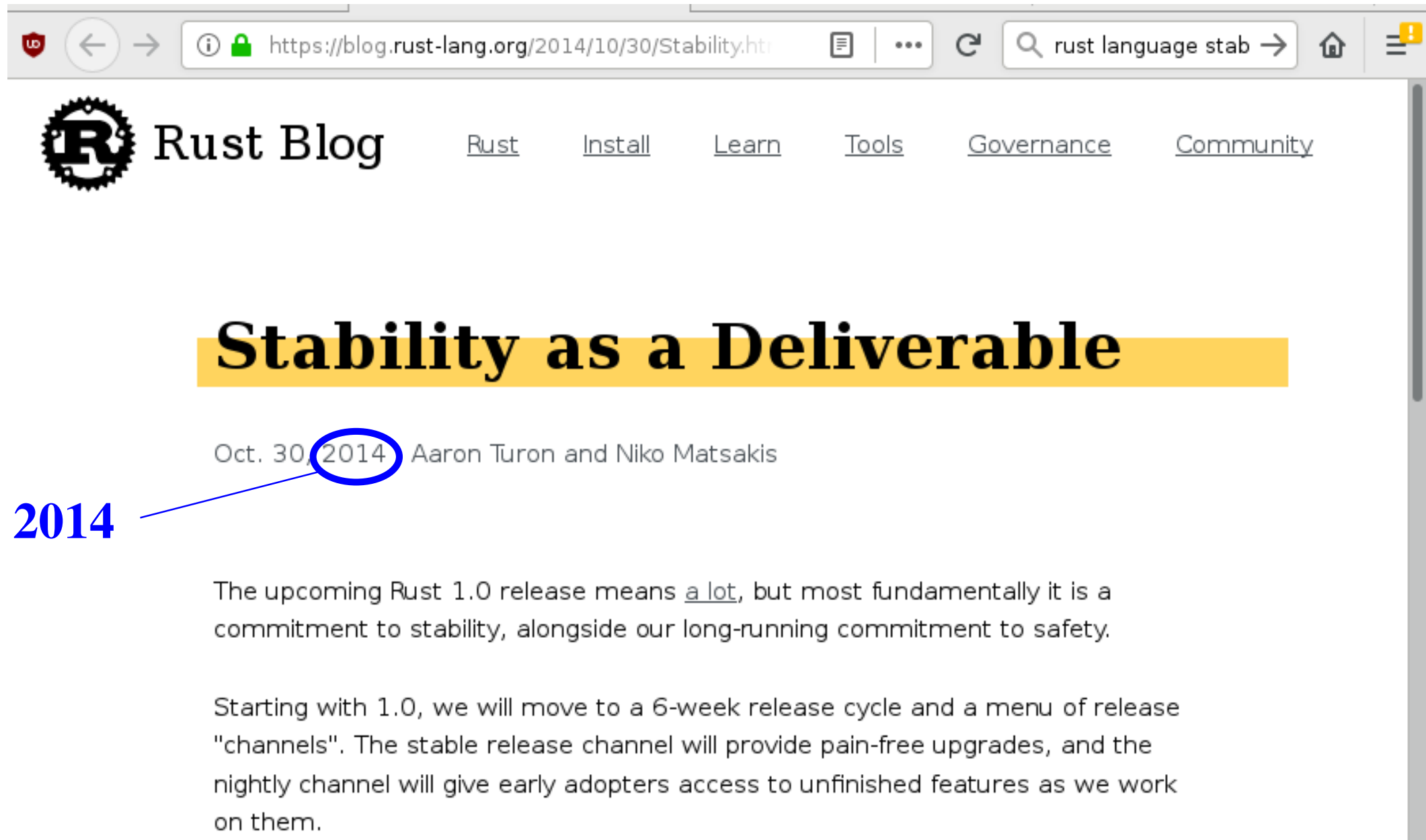
How Rust Helps You Prevent Bugs

Introduction

If you have ever written a program of any scale before, you may have run into bugs. Tiny

My personal stance is that a programming language and its implementation should strive to catch as much mistakes made by the programmer as possible, thus allow them to build better and more secure software. Although static typing makes a language more complex

Community attitude – stability



The image is a screenshot of a web browser displaying the Rust Blog. The browser's address bar shows the URL `https://blog.rust-lang.org/2014/10/30/Stability.html`. The page header includes the Rust logo and the text "Rust Blog", followed by navigation links: [Rust](#), [Install](#), [Learn](#), [Tools](#), [Governance](#), and [Community](#). The main heading of the article is "Stability as a Deliverable", which is highlighted with a yellow background. Below the heading, the date "Oct. 30, 2014" is circled in blue, and a blue line connects it to the year "2014" written in large blue font on the left side of the image. The author information "Aaron Turon and Niko Matsakis" follows the date. The article text begins with "The upcoming Rust 1.0 release means a lot, but most fundamentally it is a commitment to stability, alongside our long-running commitment to safety." and continues with "Starting with 1.0, we will move to a 6-week release cycle and a menu of release 'channels'. The stable release channel will provide pain-free upgrades, and the nightly channel will give early adopters access to unfinished features as we work on them."

Oct. 30, 2014 Aaron Turon and Niko Matsakis

Stability as a Deliverable

2014

The upcoming Rust 1.0 release means a lot, but most fundamentally it is a commitment to stability, alongside our long-running commitment to safety.

Starting with 1.0, we will move to a 6-week release cycle and a menu of release "channels". The stable release channel will provide pain-free upgrades, and the nightly channel will give early adopters access to unfinished features as we work on them.

Community attitude – error messages

```
fn main() {
    let s = String::from("hello");
    change(&s);
    println!("{}", s);
}

fn change(some_string: &String) {
    some_string.push_str(", world");
}
```

```
error[E0596]: cannot borrow immutable borrowed content
              `*some_string` as mutable
```

```
--> error.rs:8:5
```

```
7 | fn change(some_string: &String) {
   |                               ----- use `&mut String` here
   |                               to make mutable
8 |     some_string.push_str(", world");
   |     ^^^^^^^^^^^^^^^ cannot borrow as mutable
```

Community attitude



*This is the first thing
you see if you click
on "community"*

[Documentation](#)

[Install](#)

[Community](#)

[Contribute](#)

The Rust Community

The Rust programming language has many qualities, but Rust's greatest strength is the community of people who come together to make working in Rust a rewarding experience.

We are committed to providing a friendly, safe and welcoming environment for all, regardless of gender, sexual orientation, disability, ethnicity, religion, or similar personal characteristic. Our [code of conduct](#) sets the standards for behavior in all official Rust forums.

If you feel you have been or are being harassed or made uncomfortable by a community member, please [contact](#) any of the [Rust Moderation Team](#) immediately. Whether you are a regular contributor or a newcomer, we care about making the community a safe space for you.

Getting Started

Borrow checker

```
fn longest(x: &str, y: &str) -> &str {  
    if x.len() > y.len() { x } else { y }  
}
```

```
1 | fn longest(x: &str, y: &str) -> &str {  
    ^ expected lifetime parameter
```

= help: this function's return type contains a borrowed value, but the signature does not say whether it is borrowed from 'x' or 'y'

*function
parameterised
by lifetime*

lifetime annotations

```
fn longest<'a>(x: &'a str, y: &'a str)  
    -> &'a str {  
    if x.len() > y.len() { x } else { y }
```


Macros

```
use serde::{Serialize, Deserialize};
```

Not a built in Rust feature.

```
#[derive(Serialize, Deserialize, Debug)]
```

```
struct Point { x: i32, y: i32, }
```

You can do this in a library!

Only Common Lisp can beat this

```
fn main() {
```

```
    let point = Point { x: 1, y: 2 };
```

```
    // Convert the Point to a JSON string.
```

```
    let j = serde_json::to_string(&point).unwrap();
```

```
    // Parse the JSON string as a Point.
```

Awesome!

```
    let p2: Point = serde_json::from_str(&j).unwrap();
```

```
macro_rules! debug {
```

example from a personal project of mine

```
    ($g:expr, $($rhs:tt)*) => {
```

```
        (
```

```
            if debugp!($g) { eprint!($($rhs)*); }
```

```
        )
```

```
    }
```

```
}
```

What a syntax for something simple!

(and this macro doesn't even always work quite right)

Proposal: Command-line config #6699

cargo
: – (

① Open ehuss opened this issue Feb 25, 2019 · 5 comments

How to use a local unpublished crate?

Ask Question



- 7 Is there a way to use a local crate myself (for development) while leaving Cargo.toml referring to crates.io so others can also build my code? – David Roundy Jun 19 '17 at 22:21
- 1 Not possible by default at the moment. You can however work on a local branch, replace Cargo.toml with local dependency references (or mixed references), and before you merge or during, revert to or keep the main Cargo.toml file. – brokenthorn Sep 17 '18 at 14:38



Package build process

Package builds must not allow Cargo to access the network when building. In particular, they must not download or check out any sources at build time. Instead, builds must use the packaged versions of crate sources, via the corresponding library crate packages, which provide a Cargo directory registry

Package builds must set \$CARGO_HOME to a directory within the package build directory, to avoid writing to the building user's home directory outside the package build directory

From: [redacted]
To: rust-lang/cargo <cargo@noreply.github.com>
Cc: Ian Jackson
Subject: Re: [rust-lang/cargo] Want way to specify alternative
leafname to replace Cargo.toml (#6715)

...

Apart from that, this filename is intentionally non-configurable

etc.

No dynamic linking yet

Architecture support difficulties eg

<https://docs.rust-embedded.org/faq.html>

If your device architecture is not there that means **rustc** doesn't support your device. It could be that LLVM doesn't support the architecture (e.g. Xtensa, ESP8266's architecture) or that LLVM's support for the architecture is not considered stable enough and has not been enabled in **rustc** (e.g. AVR, the architecture most commonly found in Arduino microcontrollers).

Registered Targets:

aarch64	- AArch64 (little endian)
aarch64_be	- AArch64 (big endian)
arm	- ARM
arm64	- ARM64 (little endian)
armeb	- ARM (big endian)
hexagon	- Hexagon
mips	- Mips
mips64	- Mips64 [experimental]
mips64el	- Mips64el [experimental]
mipsel	- Mipsel
msp430	- MSP430 [experimental]
nvptx	- NVIDIA PTX 32-bit
nvptx64	- NVIDIA PTX 64-bit
ppc32	- PowerPC 32
ppc64	- PowerPC 64
ppc64le	- PowerPC 64 LE
riscv32	- 32-bit RISC-V
riscv64	- 64-bit RISC-V
sparc	- Sparc
sparcel	- Sparc LE
sparcv9	- Sparc V9
systemz	- SystemZ
thumb	- Thumb
thumbeb	- Thumb (big endian)
wasm32	- WebAssembly 32-bit
wasm64	- WebAssembly 64-bit
x86	- 32-bit X86: Pentium-Pro and above
x86-64	- 64-bit X86: EM64T and AMD64

Rust

The most exciting new programming language for years

`https://doc.rust-lang.org/`

Ian Jackson GNU/Citrix/Xen/Debian September 2019

`ijackson@chiark.greenend.org.uk`

C/C++? Use Rust instead if dynamic linking not needed.

Perl, Python, Haskell, Ocaml? Consider Rust.

Tcl? Do your extensions in Rust.

asm? Do the rest in Rust.

JavaScript, Java? Wish you could use Rust.

Common Lisp? OK, stick with that.

Copyright Ian Jackson / Citrix 2019

Some examples taken from the Rust Book etc, Apache 2.0 / MIT

CC-BY-SA 4.0+