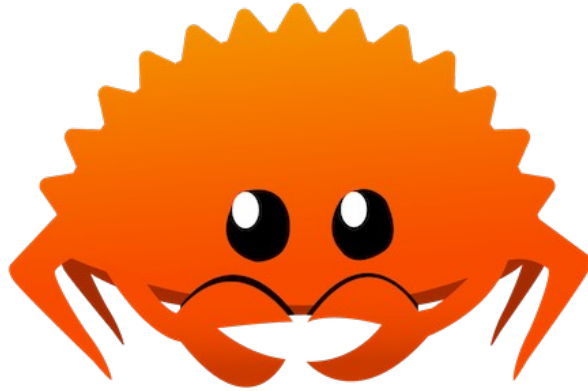


Have a crack at Rust



Tom Ryder

tom@sanctum.geek.nz

<https://sanctum.geek.nz/>

\$ whoami

- GNU/Linux systems administrator
- Former web developer
- **Free software** enjoyer
- Not a **Real Programmer**
- **Loves:** shell script, AWK, Perl
- **Accepts:** JavaScript, PHP, Python
- **Reads:** lots of C, C++
- **Writes:** a little C, no C++



Email: tom@sanctum.geek.nz
Web: <https://sanctum.geek.nz/>
Fedi: [@tejr@mastodon.sdf.org](https://mstdn.social/@tejr)

Are you at the right talk?—1/4

- I'm *not a Rust expert*.
- I'm still getting to grips with it.
- My code at work:
 - **Glue**: Make the web server do the thing with the mail server.
 - **Ephemeral**: Delete every PNG file more than 90 days old.
 - **Reports/alerts**: Email me when two lists don't match.
- I do lots of **scripting**, but not much **programming**...



“A **script** is what you give the actors.
A **program** is what you give the audience.”

—Larry Wall, [perlfaq1](#)



Are you at the right talk?—2/4

- I assume some programming knowledge, but doesn't have to be Rust.
- If you know C, C++, C#, or Java, all of this will be a **piece of cake** for you.
- If you know JavaScript or Python, you'll **get by**.
- If you know Rust, you probably know it **better than me** already.



Are you at the right talk?—3/4

- If you're **new to Rust**:
 - This talk can be to orient you to see what my own experiences have been.
 - You can get some idea of if you care or not.
 - You can *ask me and the room questions* at the end.



Are you at the right talk?—4/4

- If you **already know Rust**:
 - This is what Rust looks like to someone for whom programming is a very *secondary* thing.
 - You can get *really mad* at me. My friends do, too.
 - You can *argue with me* at the end.



Show of hands

Who's used this “Rust” thing before?



What?—1/3

- **Rust** is a statically and strongly typed general-purpose compiled programming language.
- Its compiler not only warns, it *won't even compile* code that allows classic classes of error.
- It enforces **ownership** of data for memory safety, using what it calls the **borrow checker**.



What?—2/3

- Not *philosophically* pure:
 - Higher-order functions
 - OOP-like behavior (structs and traits)
 - Algebraic data types
 - Pattern matching
- No **garbage collector**: borrow checker tracks data “lifetimes”, and enforces memory safety that way.



What?—3/3

- Started at Mozilla in 2006
- Now owned by the [Rust Foundation](#)
- Replacement for C and C++ programs
- Bindings for existing C code
- Some Rust code now in Linux, the kernel
 - This has been more than a little contentious...



Why?—1/5

- I'm a systems administrator.
- Reliability matters to me *enormously*.
 - A program can be *slow*.
 - But it better not *crash*.
- Segmentation faults and undefined behavior are my *bane* in C—*other people's* code.



Why?—2/5

```
~$ date
```

```
Mon 20 Oct 2025 04:21:02 NZDT
```

```
~$ /usr/local/bin/critical-server --please-work
```

```
Segmentation fault
```

```
~<139>$ :(█
```



Why?—3/5

- I also want programs to be *easy to install* and *portable*. I don't want to have to build it.
- If I *do* have to build it, I want the **dependencies** to be manageable.
- Compiled languages where I get just one binary with a shared library or two are ideal.



Why?—4/5

- While you guys, the real programmers, are debating **lambda functions**, **algebraic types**, **category theory**, and **trait inheritance**...
- I just want the webserver not to **crash** and the pager to wake me up at 4:00am.



Why?—5/5

- So, there's a compiled language that's:
 - as fast as C...
 - as low-level as C...
 - without C's historical baggage...
 - where C's classic memory bugs are *impossible*?





Getting started—rustup—1/2

Windows:

```
rustup-init.exe
```

***/Linux, *BSD, macOS:**

```
$ curl --proto '=https' --tlsv1.2 -sSf \  
  https://sh.rustup.rs | sh
```



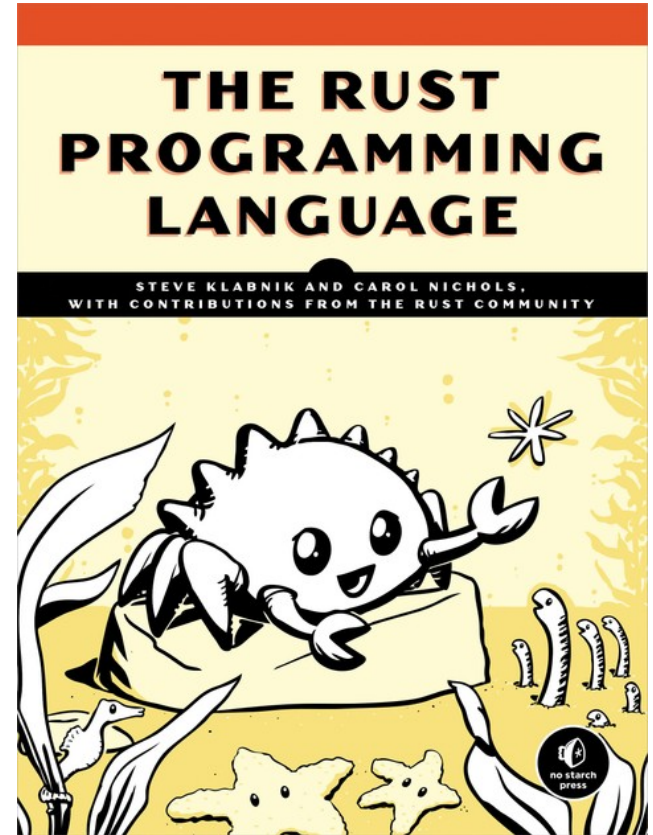
Getting started—rustup—2/2

- **rustc**, the compiler
- **cargo**, the package manager and build tool
- **clippy**, the linter
- **rustfmt**, the tidier
- The language's **documentation** (HTML)
- ***The Rust Programming Language*** book (HTML)



The Rust Programming Language

- **Free**, included with rustup
- Assumes programming knowledge
- Seems comprehensive
- Harder than **K&R**, easier than **SICP**
- Merits **persistence**:
 - It's something of a slog up to Chapter 12, where you build a little command-line application.



Hello, world!

```
main.rs + (~/projects/hello_world) - VIM
```

```
fn main() {  
    println!("Hello, world!");  
}
```



Hello, cargo!

```
~/projects
```

```
~/projects$ cargo new hello
```

```
    Creating binary (application) `hello` package
```

```
note: see more `Cargo.toml` keys and their definitions at https://doc.rust-lang.org/cargo/reference/cargo-toml.html
```

```
~/projects$ tree hello
```

```
hello
```

```
├── Cargo.toml
```

```
└── src
```

```
    └── main.rs
```

```
2 directories, 2 files
```

```
~/projects$ █
```



Directions

- I won't demonstrate Rust syntax from first principles.
- I also won't guide you through **ownership** and the **borrow checker**, which is a bit painful for a presentation.
 - The book does a good job of that, anyway.
- Instead, I'll demonstrate a few features that stood out to me, and that have examples of Rust's approach to safety.



Enums—1/6

main.rs + (~/projects/temperature/src) - VIM

```
enum TemperatureUnit {  
    Celsius,  
    Fahrenheit,  
}  
  
***  
let number: f64;  
let unit: TemperatureUnit;  
***  
match unit {  
    TemperatureUnit::Celsius => {  
        let fahrenheit: f64 = (number * (9.0/5.0)) + 32.0;  
    }  
    TemperatureUnit::Fahrenheit => {  
        let celsius: f64 = (number - 32.0) / (9.0/5.0);  
    }  
}
```



Enums—2/6

- That's not really special so far...just like a `case` block in C, or a `match` block in Python v3.10+.
- However, you can **associate** values with enums, too, and extract them in the `match` block.
- Very useful!
- You might use **unions** in C for something similar.



Enums—3/6

main.rs + (~/.projects/temperature/src) - VIM

```
enum Temperature {  
    Celsius(f64),  
    Fahrenheit(f64),  
}  
  
***  
let temperature: Temperature;  
  
***  
let converted = match temperature {  
    Temperature::Celsius(number) =>  
        Temperature::Fahrenheit((number * (9.0/5.0)) + 32.0),  
    Temperature::Fahrenheit(number) =>  
        Temperature::Celsius((number - 32.0) / (9.0/5.0)),  
};
```

Enums—4/6

- Enum matching is **exhaustive**.
- If you don't handle a possible value, the compiler rejects the code.
- This is a good example of the compiler enforcing **safety**.
- Also a good example of the high-quality error output `rustc` provides:



Enums—5/6

main.rs = (~/projects/temperature/src) - VIM

```
enum Temperature {  
    Celsius(f64),  
    Fahrenheit(f64),  
    Kelvin(f64),  
}
```



Enums—6/6

```
~/projects/temperature/src
~/projects/temperature/src(master+?)$ rustc main.rs
error[E0004]: non-exhaustive patterns: `Temperature::Kelvin(_)` not covered
  -> main.rs:10:27
   |
10 |     let converted = match temperature {
   |                        ~~~~~ pattern `Temperature::Kelvin(_)` not covered
   |
note: `Temperature` defined here
  -> main.rs:1:6
   |
 1 | enum Temperature {
   |     ~~~~~
   |
...
 4 |     Kelvin(f64),
   |     ~~~~~ not covered
   = note: the matched value is of type `Temperature`
help: ensure that all possible cases are being handled by adding a match arm with a wildcard
   |
14 ~         Temperature::Celsius((number - 32.0) / (9.0/5.0)),
15 ~         Temperature::Kelvin(_) => todo!(),
   |

error: aborting due to 1 previous error

For more information about this error, try `rustc --explain E0004`.
~/projects/temperature/src(master+?)<1>$
```



Structs

main.rs + (~/.rust/minigrep/src) - VIM

```
struct Config {  
    pub query: String,  
    pub file_path: String,  
    pub ignore_case: bool,  
}
```



Implementations—1/2

- You don't declare methods for structs in the body.
- Instead you write a separate **implementation** block (`impl`), implementing the methods to run on it.
- Not *quite* OOP—certainly not purist—but it does the important things.



Implementations—2/2

presentation.rs + (~/projects) - VIM

```
struct Presentation {  
    title: String,  
    author: String,  
}  
  
impl Presentation {  
    fn summarize(&self) -> String {  
        format!("{}", by {}, self.title, self.author)  
    }  
}
```



Traits—1/2

- You can specify a set of methods as **traits** for a struct to support.
- You can then use the traits in the type system.
- Instead of asserting “is a Presentation”, you can assert “implements Summary”.
- **Duck typing**: Typing based on what something *does*, rather than what it *is*.



Traits—2/2

presentation.rs + (~/projects) - VIM

```
trait Summary {  
    fn summarize(&self) -> String;  
}  
  
impl Summary for Presentation {  
    fn summarize(&self) -> String {  
        format!("{}", by {}, self.title, self.author)  
    }  
}
```



Error handling—1/5

- There are no **exceptions** in Rust.
- To start with error handling, you can just call the `panic!()` macro, which simply exits the program immediately.
- Think of it like a `die()`.



Error handling—2/5

panic.rs (~/projects) - VIM

```
fn division(dividend: i32, divisor: i32) -> i32 {  
    if divisor == 0 {  
        panic!("division by zero");  
    } else {  
        dividend / divisor  
    }  
}
```



Error handling—3/5

- More comprehensive and a very common pattern is to use the `Result<T, E>` generic type.
- If what you did worked, return the value in type `T`.
- If there was an error, return an error in the type `E`.
- The compiler enforces handling all cases.



Error handling—4/5

handle.rs + (~/.rust) - VIM

```
pub fn div(x: f64, y: f64) -> Result<f64, String> {  
    if y == 0.0 {  
        Err(String::from("division by zero"))  
    } else {  
        Ok(x / y)  
    }  
}
```



Error handling—5/5

- Some of the function names chosen here aren't great.
- To specify that you want to panic on an error result, you use **.expect(message)**. *Wat?*
- To specify that you want to pass the result through a method chain, but panic on an error result, you use **.unwrap()**. *Wat?*
- Can anyone think of better names?



Collections—1/3

- **Arrays** are of fixed length, like in C. They must have the *same* type.
- **Vectors** have a dynamic length list of data, also all of the *same* type.



Collections—2/3

- **Tuples** can collect *different* types in the same order, in fixed length.
- **HashMaps** uniquely index a list of data of one type with data of another (“dictionaries”, “associative arrays”).



Collections—3/3

- Borrow checking and ownership are particularly curly with collections.
- If you pass non-Copy pieces of the collection around, the borrow checker will scold you.
- You will be tempted to `clone()` everything...
- The saving grace is the compiler is very helpful in suggesting fixes.



Libraries

- I've had no problems finding **crates** (packages) with libraries for what I need.
- Crates for typical tasks at work might be:
 - MariaDB, MySQL, PostgreSQL: `sqlx`
 - systemd interfacing: `systemd`
 - JSON/XML parsing: `json`, `xml-rs`



Tooling—Compiler

- In `rustc`, the compiler's warning and error output is *really, really good*.
- It's easily the most helpful I've seen in any programming language.
- Writing Rust with its strictures can be slow, but trusting in the compiler for a *quick* development cycle really helps.
- I feel confident in coding fast, and waiting for the compiler to correct me.



Tooling—Linter—1/2

- Run `cargo clippy` for “lint” output.
- Stuff that isn’t errors or even warnings
 - The compiler gives you plenty of those, anyway.
- Things like unneeded keywords, or suggesting syntactic sugar.
- I like static analysis and linters, so I like getting a good one in the `rustup` box.



Tooling—Linter—2/2

~/rust/rbdh

~/rust/rbdh\$ cargo clippy

warning: redundant field names in struct initialization

→ src/main.rs:16:29

```
16 |         Self {matcher: reg, printstr: printstr}  
   |                               help: replace it with: `printstr`
```

= **help:** for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#redundant_field_names

= **note:** `#[warn(clippy::redundant\_field\_names)]` on by default

warning: unneeded `return` statement

→ src/main.rs:24:9

```
24 |         return false;
```

= **help:** for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#needless_return

= **note:** `#[warn(clippy::needless\_return)]` on by default

help: remove `return`

```
24 -         return false;
```

```
24 +         false
```

Tooling—Formatter

- Just use `rustfmt`, included with `rustup`.
- I like code being formatted *consistently*, but I don't go much in for holy wars.
 - Tabs vs spaces, method on *this* line vs the *next* one, maximum line length...
- I haven't been angry with anything `rustfmt` has done.
- I use it in Vim and have had no problems.



Adoption—1/4

- My favourite example of the strains of Rust adoption into existing codebases has been from the Python cryptography module.
- Since cryptography v35.0, Rust has been needed to build from source.
- People were [very upset about this](#), as it added Rust as a non-trivial dependency, complicating packaging and build workflow.
- The maintainer did not budge.



Adoption—2/4



tiran on Feb 8, 2021

Contributor ...

It is also impossible to build on ANY alpine linux version/architecture.

Update to most recent Alpine with Rust ≥ 1.45 . Alpine latest has rust (1.47.0-r2).

Please remove this dependency as soon as possible.

Not going to happen. Rust dependency will stay to replace C code with an actual safe language.

You can disable Rust integration in 3.4.x (see FAQ). Starting with 3.5 cryptography will have mandatory Rust code.

 118

 37

 5

 41



Adoption—3/4

- Adoption into the Linux kernel has been similarly fraught.
- From observations on the kernel mailing list, I would say the number-one thing Rust advocates lack is *patience*.
- Tempers flared whenever progress getting Rust into the kernel stalled.
- Rust isn't going to save our souls, and using C is not a cardinal sin.



Adoption—4/4

```
On Wed, Jan 08, 2025 at 04:16:18PM +0100, Miguel Ojeda wrote:  
> On Wed, Jan 8, 2025 at 3:00 PM Christoph Hellwig <hch@lst.de> wrote:  
> >  
> > No rust code in kernel/dma, please.  
>  
> What do you suggest?
```

Keep the wrappers in your code instead of making life painful for others.



Tooling—IDE

- I don't use an IDE. I'm a shell-and-Vim guy.
- I've had **no problems** using Vim with Rust.
- Vim's built-in support works great.
- There are LSP options for comprehensive support, but I don't really like that.
- The compiler and linter work fine called straight from Vim.



Overall—1/6—Advantages

- Very **rewarding** to know.
- Very **instructive** about problem patterns in other programming languages.
 - Even when you're not using it.
- Very practical **feature choices**.
 - Reminds me of Perl: give me all the *useful stuff*.
 - Screw design purity!



Overall—2/6—Advantages

- Really good **warning and error messages**.
 - The best of any language I've used.
 - This alone is a reason to try Rust, even if you don't end up using it much.
- Great to have **Unicode** built-in to strings.
- Wonderful to have a whole class of errors completely disappear. Can be a good tradeoff.



Overall—3/6—Drawbacks

- **Hardest language** I've ever tried to learn.
- **Unintuitive, needlessly strange** design and syntax.
 - Unmatched single quotes for lifetimes...?
- Some awful choices of **function names** in the stdlib.
- Dealing with **strings** even *more* confusing than in C.
 - The book tries to explain this away, but it's not very convincing.
 - At least it's more **explicit** about the curliness than C.



Overall—4/6—Drawbacks

- Still a **heavy dependency** to install on many machines, if building is required.
 - If I can build a binary for the target architecture, though, no problems.
- **Overzealous** and sometimes downright **sanctimonious** community.
 - A little too much non-software politics for my liking, too.
- **Anti-copyleft** culture.



Overall—5/6—Conclusions

- I didn't like it at first, but I have come around.
- Its largely replacing C++ seems inevitable, but it will take a **long time**.
- I think it'll replace some C, too, but not as much.
- It will be **useful**. If new code needs C or C++ from now on, I will probably write it in Rust.



Overall—6/6—Conclusions

- But I don't think I'll be using it in **day-to-day-scripting**.
 - Too slow to write.
 - Too strict.
 - Not already-installed on any given machine.
 - Efficiency at runtime a bad trade-off for quick jobs.
- I'll just keep using Perl, PHP, Python, and shell for those.
- So... what next?



You swap places with Slinky.

```
-----  
| .....|  
#+.....|  
#|.....|  
#|.....|  
#|.....|  
#-----  
###  
#  
T  
B#
```

[Tejr the Footpad] St:16 Dx:18 Co:15 In:11 Wi:11 Ch:8 Chaotic
Dlv1:1 \$:0 HP:11(11) Pw:2(2) AC:7 Xp:1

Plug for PLUG

- I present regularly at the **Palmerston North Linux Users Group**:
<https://www.plug.org.nz/>
- We meet once a month, every second Wednesday.
- Everyone welcome!



Questions? Arguments?

- Rust Foundation
- Rustlings

Email: tom@sanctum.geek.nz

Website: <https://sanctum.geek.nz/>

Fediverse: [@tejra@mastodon.sdf.org](https://mstdn.social/@tejra)



Thanks to: Gareth Pulham, Daniel Foster, Robbie McKennie