Open Source Development with Rust

Orhun Parmaksız

Who am I?

- Open Source Developer
  - https://github.com/orhun
- GNU/Linux user, member of the Arch Linux team
  - https://archlinux.org/people/trusted-users/#orhun
- <insert job title>
  - https://www.linkedin.com/in/orhunp
- <dank memes>
  - https://twitter.com/orhunp_

https://orhun.dev

A highly customizable changelog generator ⛰️

https://git-cliff.org

Roadmap

- Introduction to Rust
- Creating our first project
- Maintenance tips
- Tools
- Q&A

Rust

Rust is a multi-paradigm programming language designed for performance and safety, especially safe concurrency.

It is syntactically similar to C++, but can guarantee memory safety (without garbage collection) by using a borrow checker to validate references.

https://www.rust-lang.org/

Features of Rust

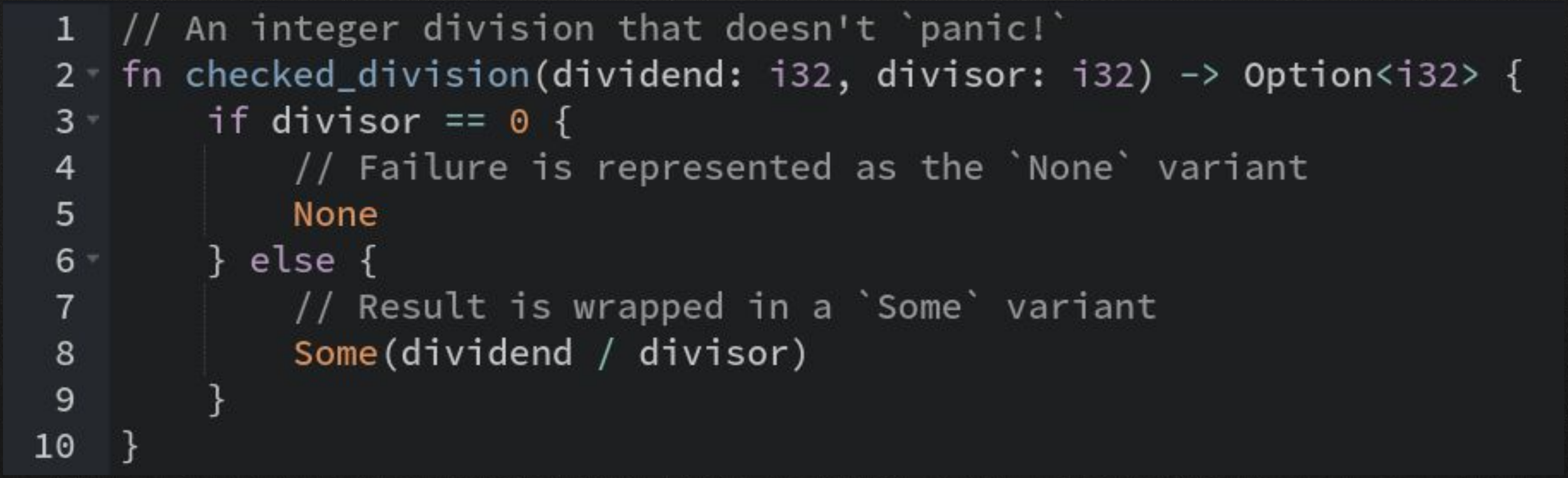
- Memory safety
  - No {null, dangling} pointers & data races
  - Option<T> type
  - Lifetime management
  - “unsafe” blocks
- Memory management
  - RAII (resource acquisition is initialization)
- Ownership
  - References
- Performance
  - Idiomatic Rust

Memory safety

Rust is designed to be memory safe, and it does not permit null pointers, dangling pointers, or data races in safe code.

Memory safety

Rust core library provides an option type, which can be used to test whether a pointer has Some value or None.

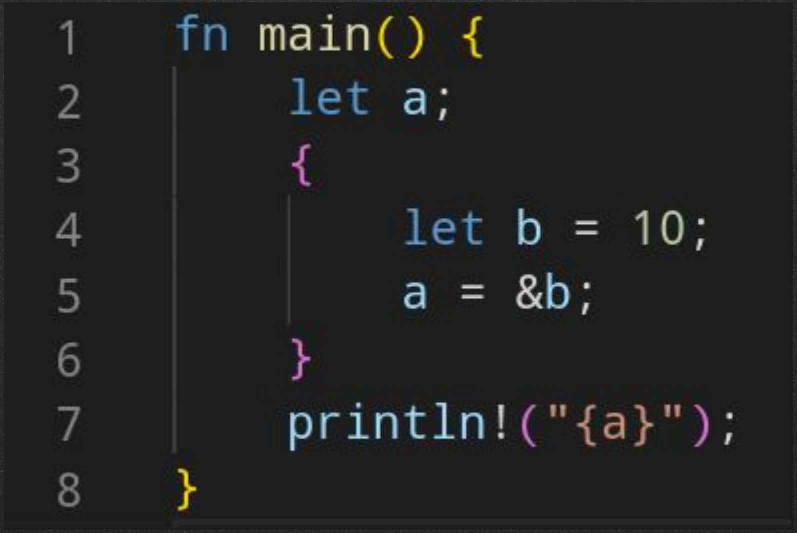


```rust
// An integer division that doesn't `panic!`
fn checked_division(dividend: i32, divisor: i32) -> Option<i32> {
    if divisor == 0 {
        // Failure is represented as the `None` variant
        None
    } else {
        // Result is wrapped in a `Some` variant
        Some(dividend / divisor)
    }
}
```

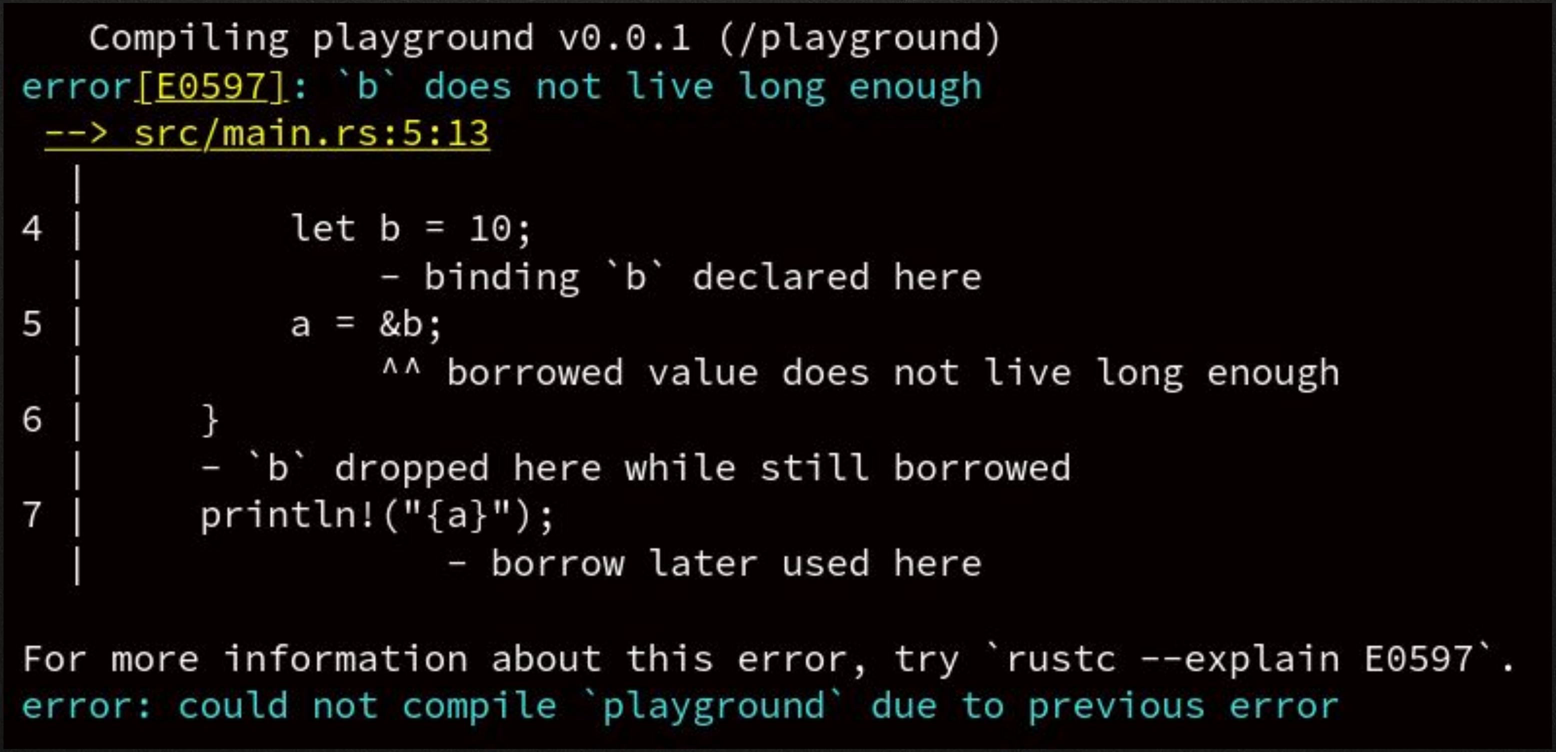
https://doc.rust-lang.org/rust-by-example/std/option.html

Memory safety

Rust introduces added syntax to manage lifetimes, and the compiler reasons about these through its borrow checker.



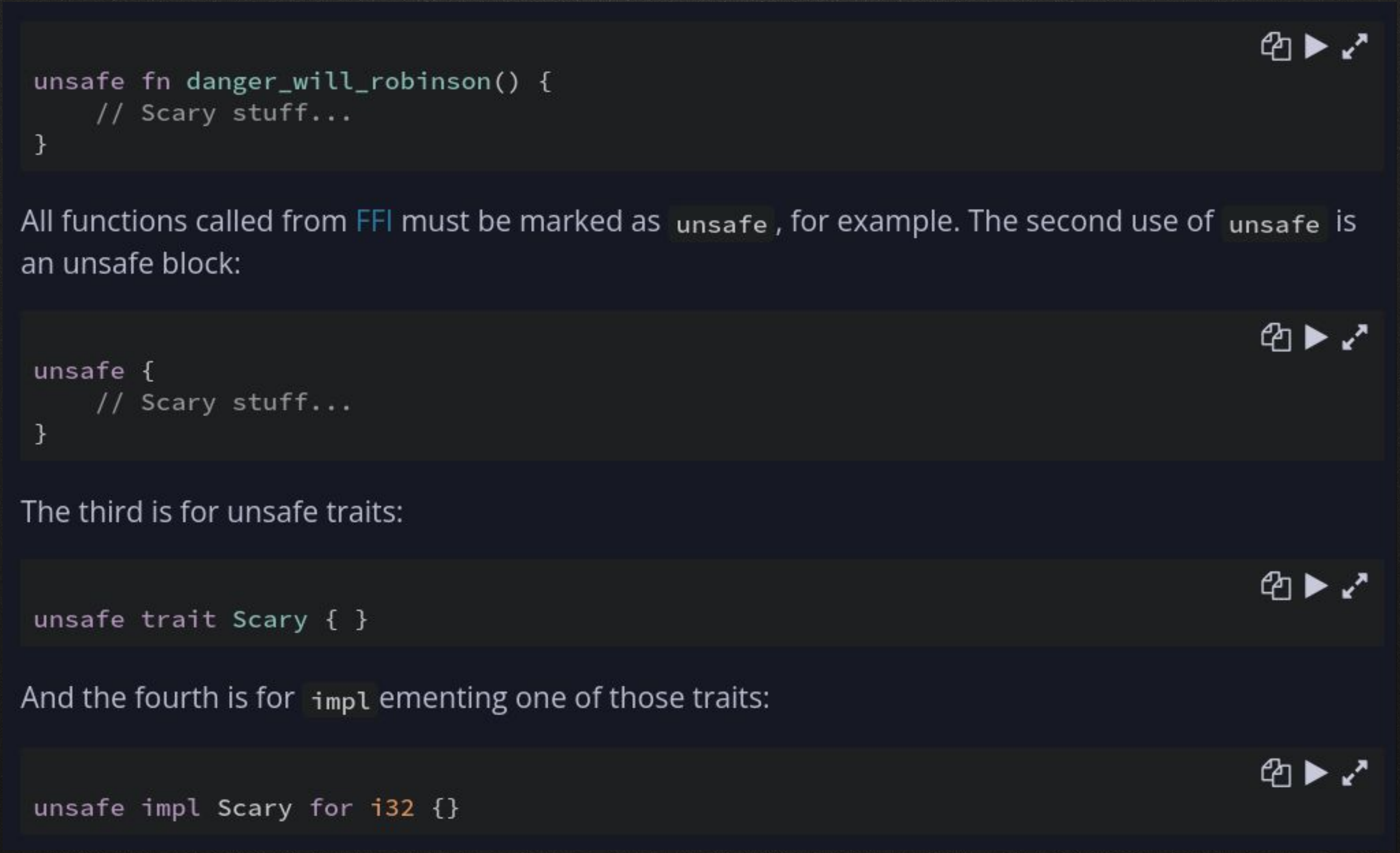
```rust
fn main() {
    let a;
    {
        let b = 10;
        a = &b;
    }
    println!("{a}");
}
```



```
   Compiling playground v0.0.1 (/playground)
error[E0597]: `b` does not live long enough
 --> src/main.rs:5:13
  |
4 |         let b = 10;
  |             - binding `b` declared here
5 |         a = &b;
  |             ^^ borrowed value does not live long enough
6 |     }
  |     - `b` dropped here while still borrowed
7 |     println!("{a}");
  |               - borrow later used here

For more information about this error, try `rustc --explain E0597`.
error: could not compile `playground` due to previous error
```

Memory safety

Unsafe code which can subvert some of these restrictions may be written using the language's unsafe keyword.



```rust
unsafe fn danger_will_robinson() {
    // Scary stuff...
}
```

All functions called from FFI must be marked as `unsafe`, for example. The second use of `unsafe` is an unsafe block:

```rust
unsafe {
    // Scary stuff...
}
```

The third is for unsafe traits:

```rust
unsafe trait Scary { }
```

And the fourth is for `impl`ementing one of those traits:

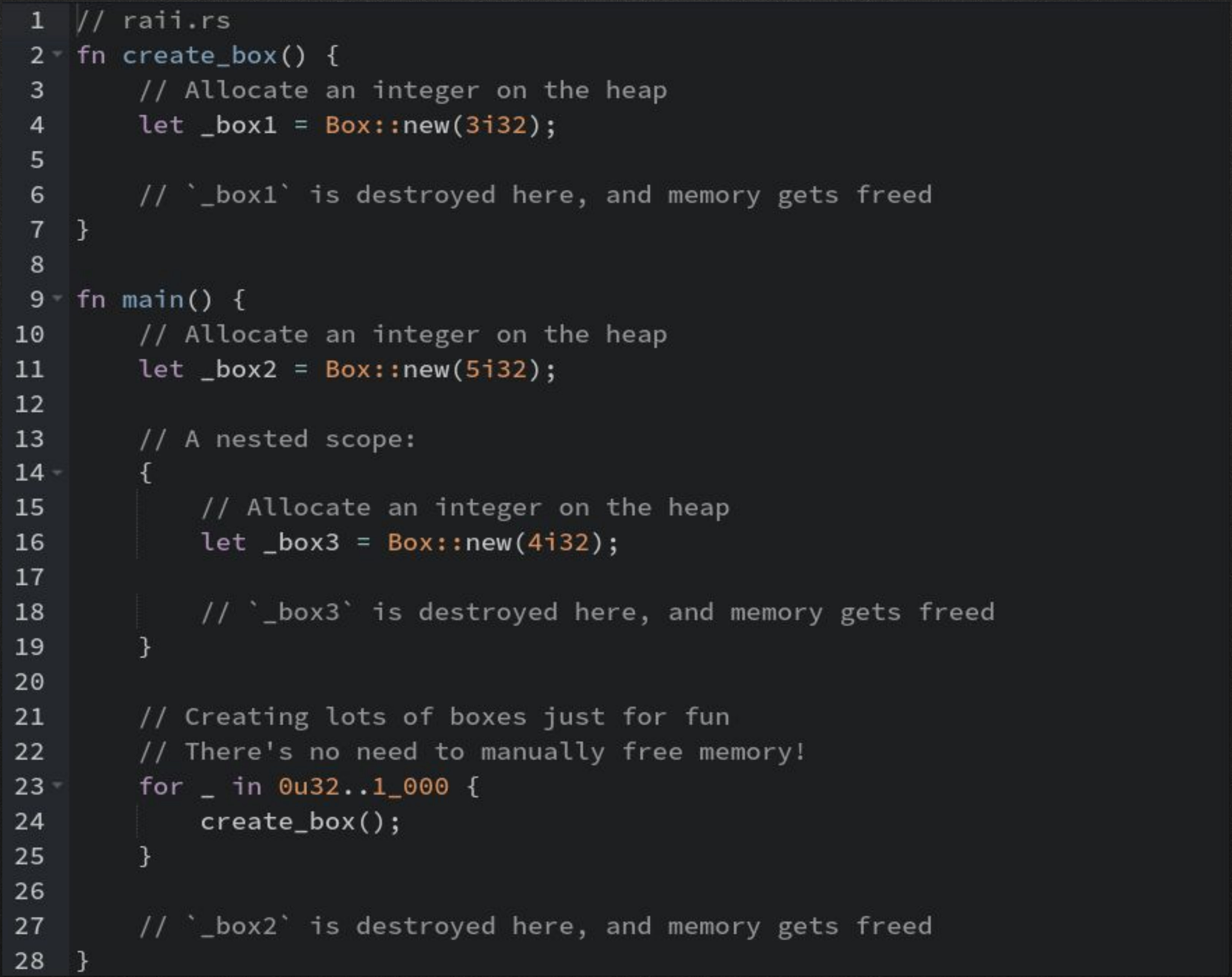
```rust
unsafe impl Scary for i32 {}
```

https://doc.rust-lang.org/1.30.0/book/first-edition/unsafe.html

Memory management

Rust does not use an automated garbage collection system. Instead, memory and other resources are managed through the resource acquisition is initialization (RAII) convention.

“Whenever an object goes out of scope, its destructor is called and its owned resources are freed.”



```rust
// raii.rs
fn create_box() {
    // Allocate an integer on the heap
    let _box1 = Box::new(3i32);

    // `_box1` is destroyed here, and memory gets freed
}

fn main() {
    // Allocate an integer on the heap
    let _box2 = Box::new(5i32);

    // A nested scope:
    {
        // Allocate an integer on the heap
        let _box3 = Box::new(4i32);

        // `_box3` is destroyed here, and memory gets freed
    }

    // Creating lots of boxes just for fun
    // There's no need to manually free memory!
    for _ in 0u32..1_000 {
        create_box();
    }

    // `_box2` is destroyed here, and memory gets freed
}
```

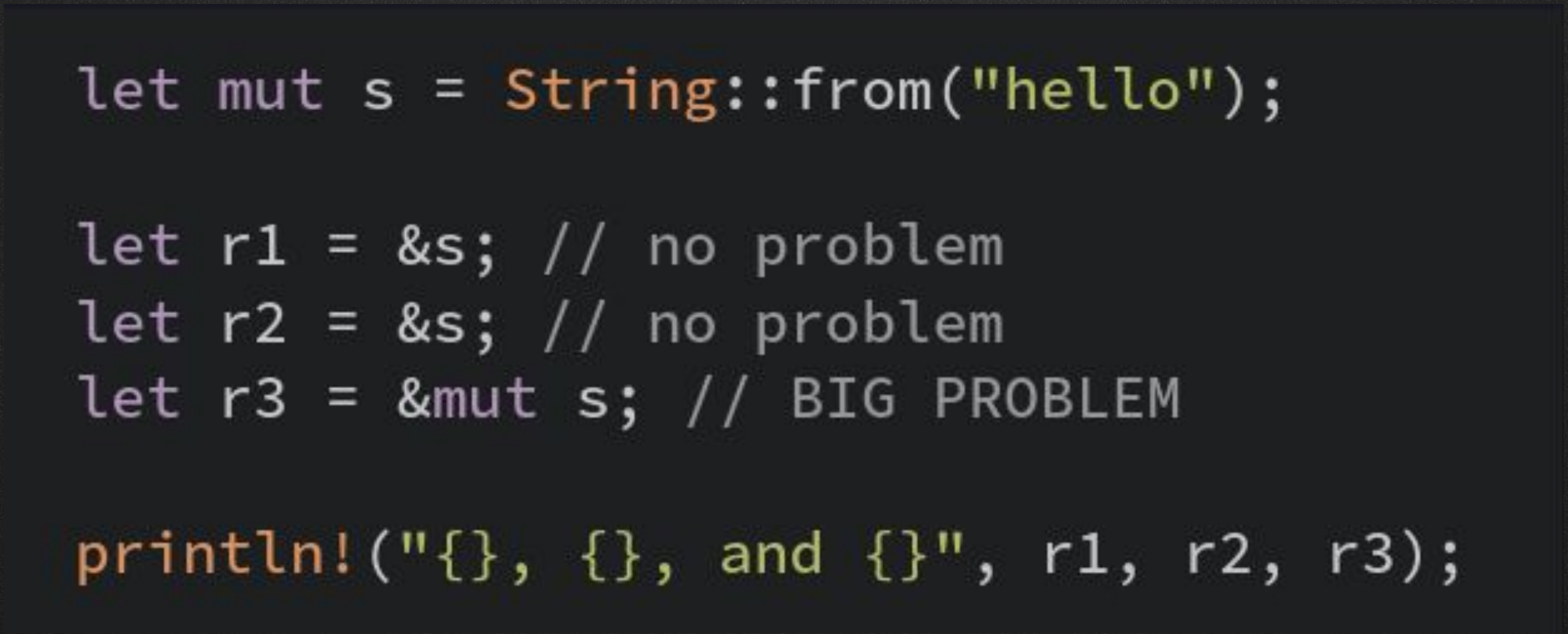
https://doc.rust-lang.org/rust-by-example/scope/raii.html

Ownership

Rust has an ownership system where all values have a unique owner, and the scope of the value is the same as the scope of the owner.

Values can be passed by immutable reference, using `&T`, by mutable reference, using `&mut T`, or by value, using `T`.

At all times, there can either be multiple immutable references or one mutable reference. The Rust compiler enforces these rules at compile time and also checks that all references are valid.

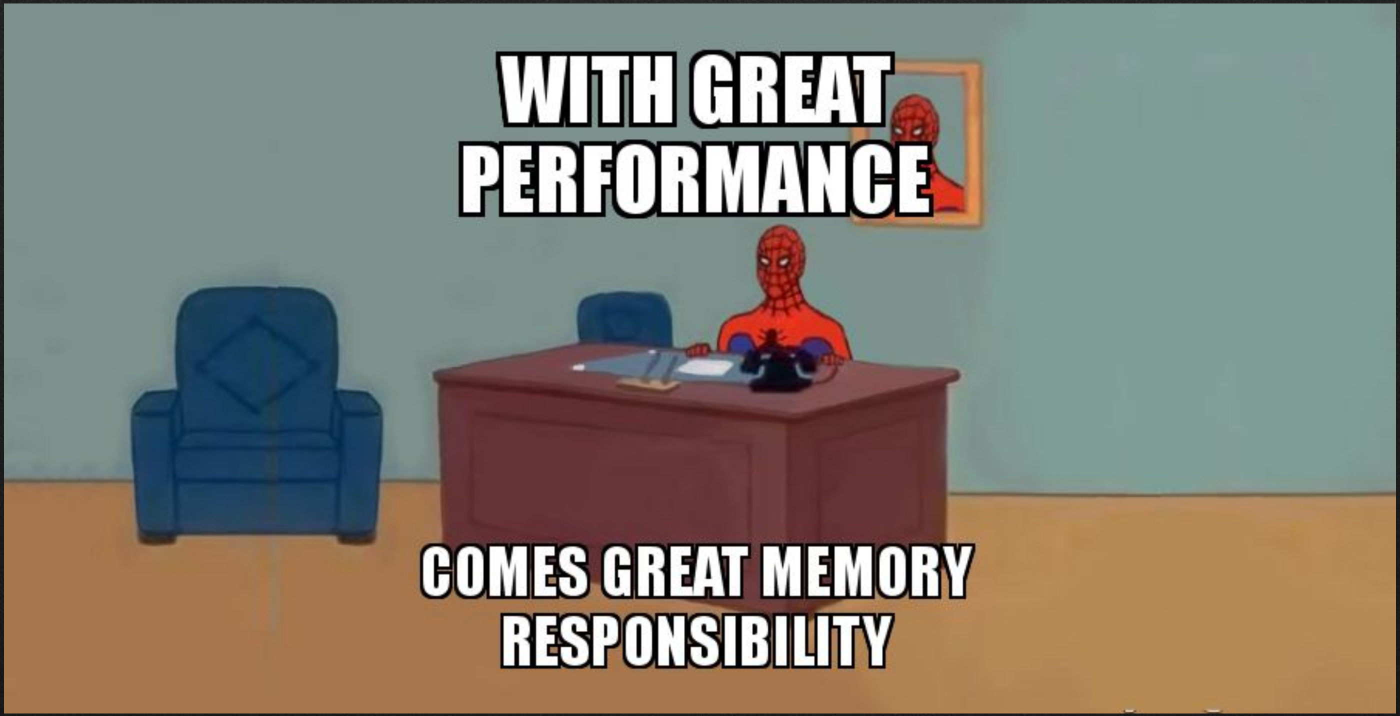


```rust
let mut s = String::from("hello");

let r1 = &s; // no problem
let r2 = &s; // no problem
let r3 = &mut s; // BIG PROBLEM

println!("{}, {}, and {}", r1, r2, r3);
```

https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html

Performance

Performance of idiomatic Rust is comparable to the performance of idiomatic C++.



Why Rust?

- Performance
  - Fast & Efficient
- Reliability
  - Memory Safety
  - Error Handling
- Productivity
  - General Purpose
  - Project Oriented
  - Well Supported

What’s bad about Rust?

- Nothing
- Steep learning curve
- Compiler enforcing rules that would be "best practices" elsewhere
- Longer compile times
- Missing platform support for some domains
- Use of unsafe can secretly break safety guarantees (The Rustonomicon)

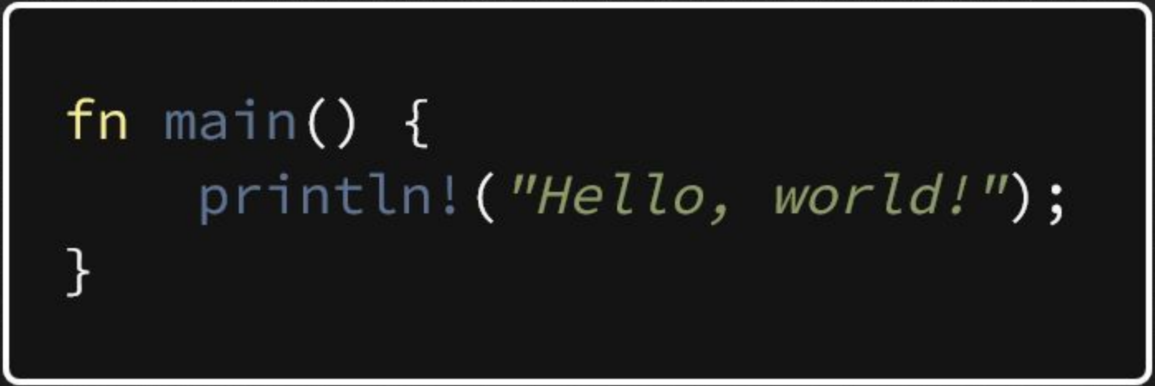
https://blog.rust-lang.org/2020/04/17/Rust-survey-2019.html#why-not-use-rust

What can we build?

- CLI
- WASM
- Networking
- Embedded

https://doc.rust-lang.org/nightly/rustc/platform-support.html

Let’s get started!



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

- https://doc.rust-lang.org/book/ch01-00-getting-started.html
- https://cheats.rs/
- https://github.com/rust-lang/rustlings

rustc

rustc is the compiler for the Rust programming language, provided by the project itself. Compilers take your source code and produce binary code, either as a library or executable.

Most Rust programmers don't invoke rustc directly, but instead do it through cargo.

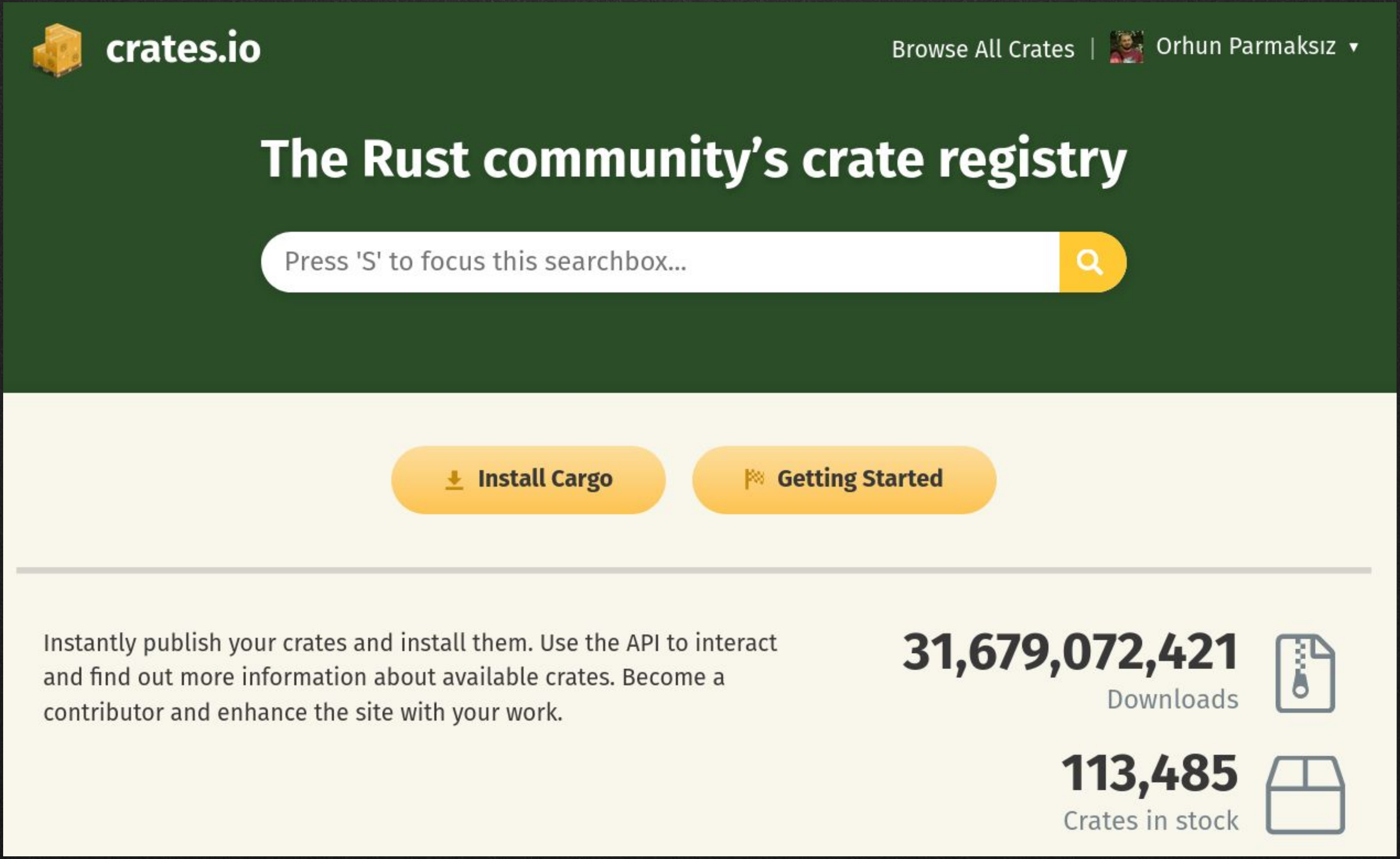
https://github.com/rust-lang/rust

cargo

Cargo is the package manager and crate host for Rust. It downloads your Rust project’s dependencies and compiles your project.

https://github.com/rust-lang/cargo

crates.io


crates.io

Browse All Crates | Orhun Parmaksız ▾

The Rust community's crate registry

Press 'S' to focus this searchbox...

Install Cargo

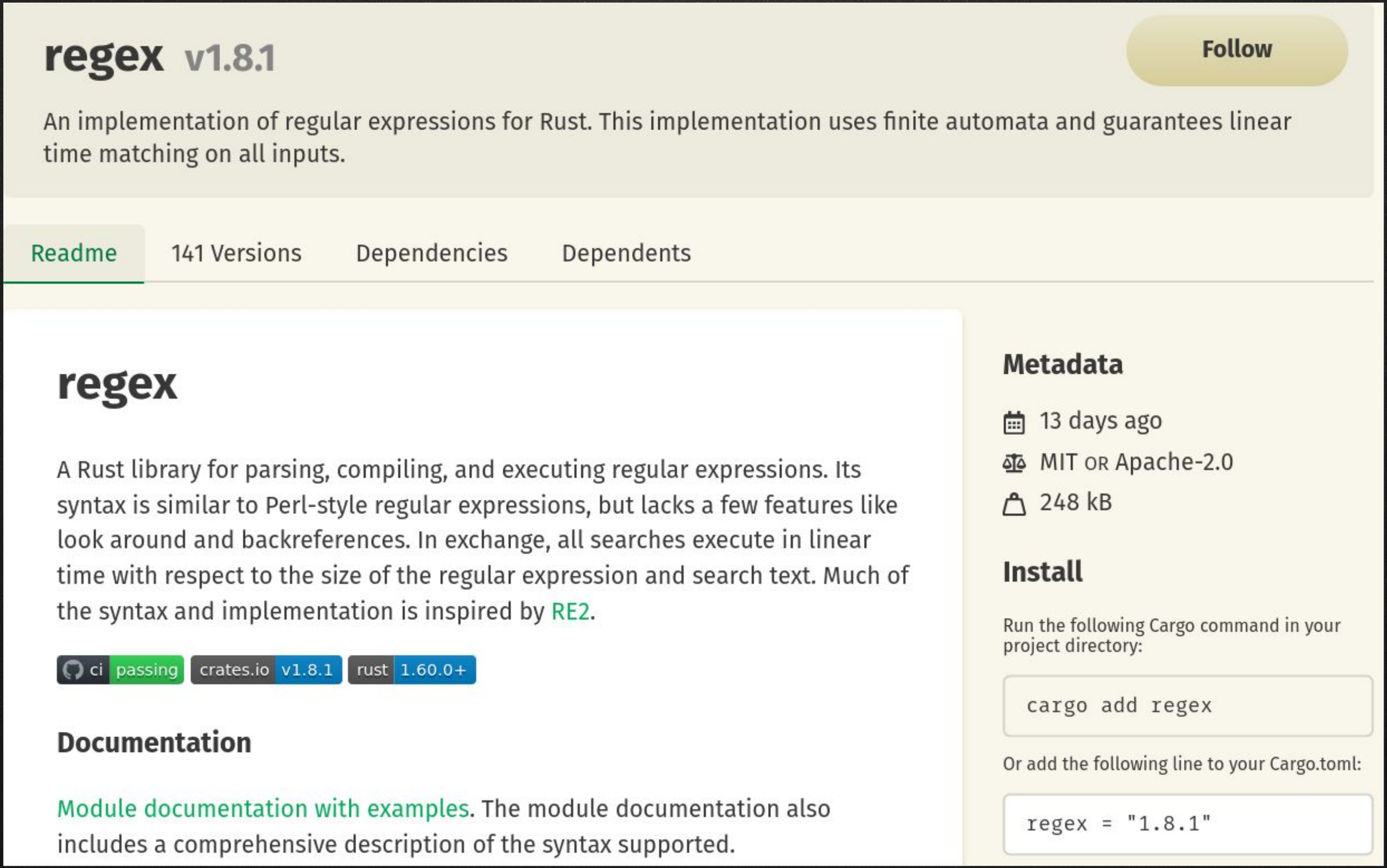
Getting Started

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

31,679,072,421
Downloads

113,485
Crates in stock

https://crates.io


regex v1.8.1

Follow

An implementation of regular expressions for Rust. This implementation uses finite automata and guarantees linear time matching on all inputs.

Readme | 141 Versions | Dependencies | Dependents

regex

A Rust library for parsing, compiling, and executing regular expressions. Its syntax is similar to Perl-style regular expressions, but lacks a few features like look around and backreferences. In exchange, all searches execute in linear time with respect to the size of the regular expression and search text. Much of the syntax and implementation is inspired by RE2.

ci passing | crates.io v1.8.1 | rust 1.60.0+

Documentation

Module documentation with examples. The module documentation also includes a comprehensive description of the syntax supported.

Metadata

13 days ago

MIT OR Apache-2.0

248 kB

Install

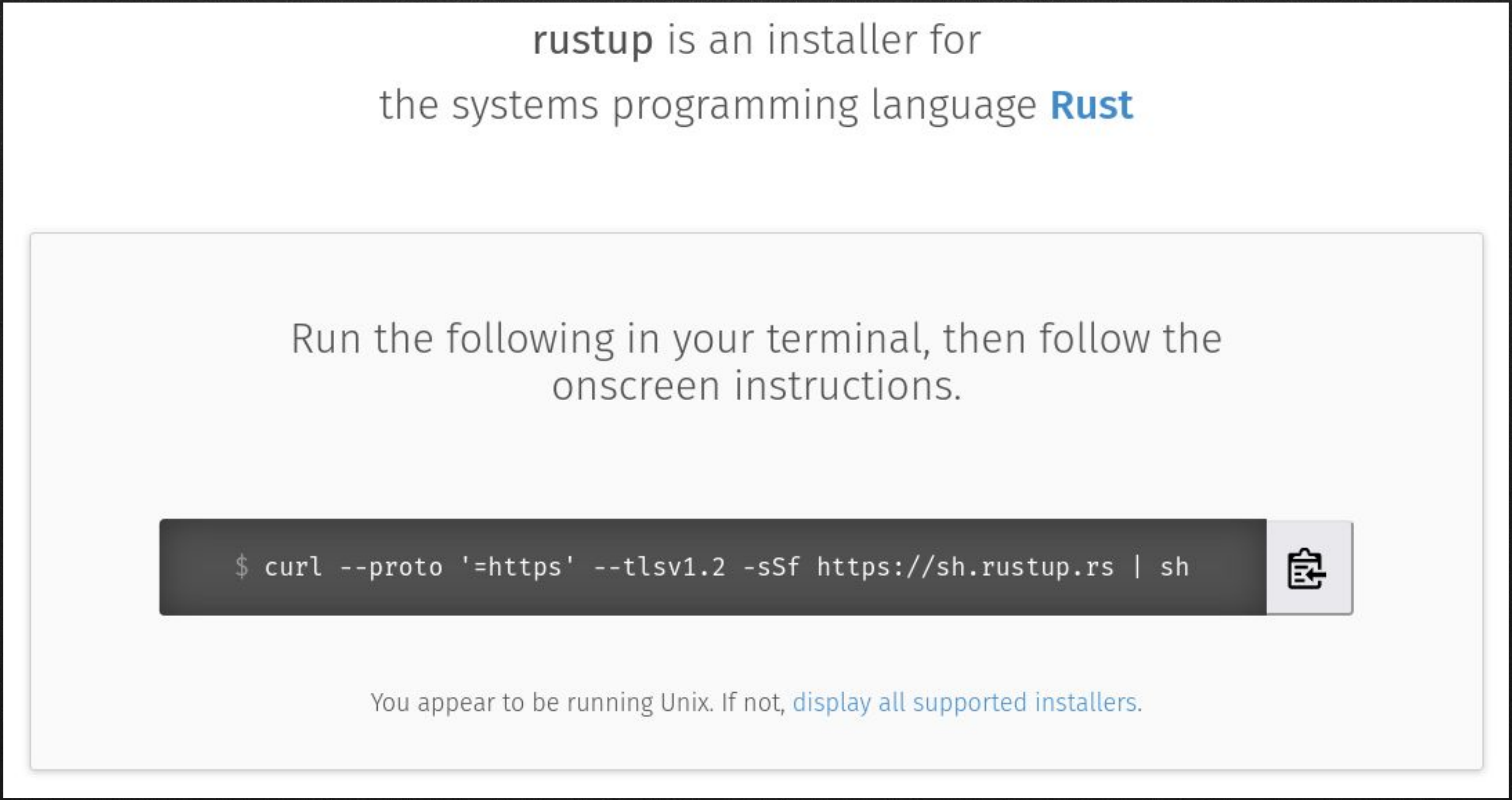
Run the following Cargo command in your project directory:

```
cargo add regex
```

Or add the following line to your Cargo.toml:

```
regex = "1.8.1"
```

rustup



rustup is an installer for
the systems programming language **Rust**

Run the following in your terminal, then follow the onscreen instructions.

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

You appear to be running Unix. If not, display all supported installers.

https://rustup.rs/

rustfmt

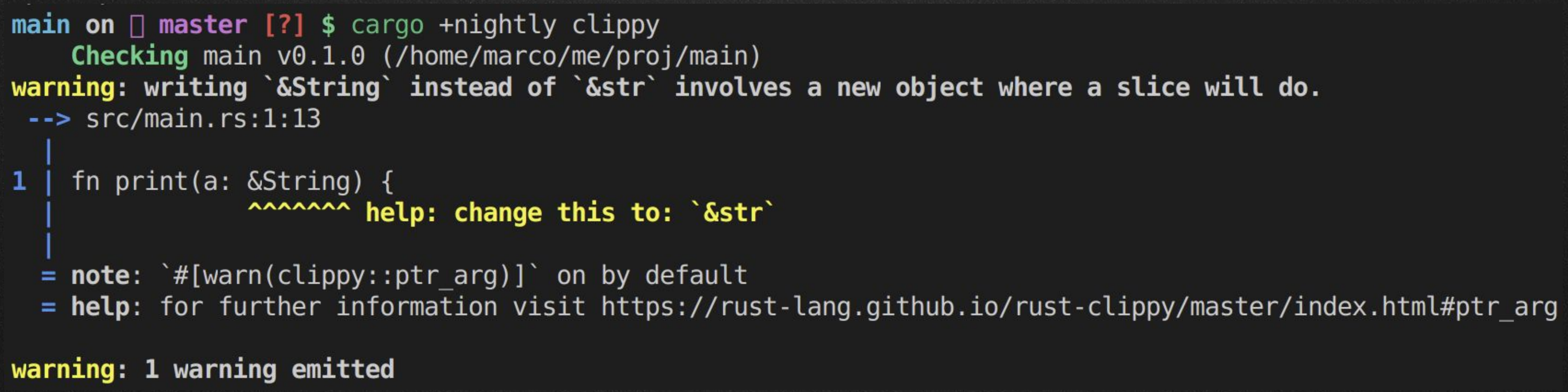
A tool for formatting Rust code according to style guidelines.

https://github.com/rust-lang/rustfmt

clippy

A collection of lints to catch common mistakes and improve your Rust code.

https://github.com/rust-lang/rust-clippy



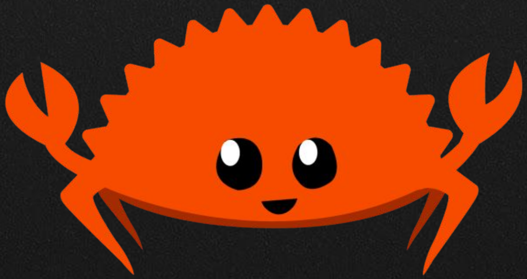
```
main on  master [?] $ cargo +nightly clippy
    Checking main v0.1.0 (/home/marco/me/proj/main)
warning: writing `&String` instead of `&str` involves a new object where a slice will do.
 --> src/main.rs:1:13
  |
1 | fn print(a: &String) {
  |             ^^^^^^^ help: change this to: `&str`
  |
  = note: `#[warn(clippy::ptr_arg)]` on by default
  = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#ptr_arg

warning: 1 warning emitted
```

Let’s create our first Rust program

Objective: print a random number and exit.

Steps:

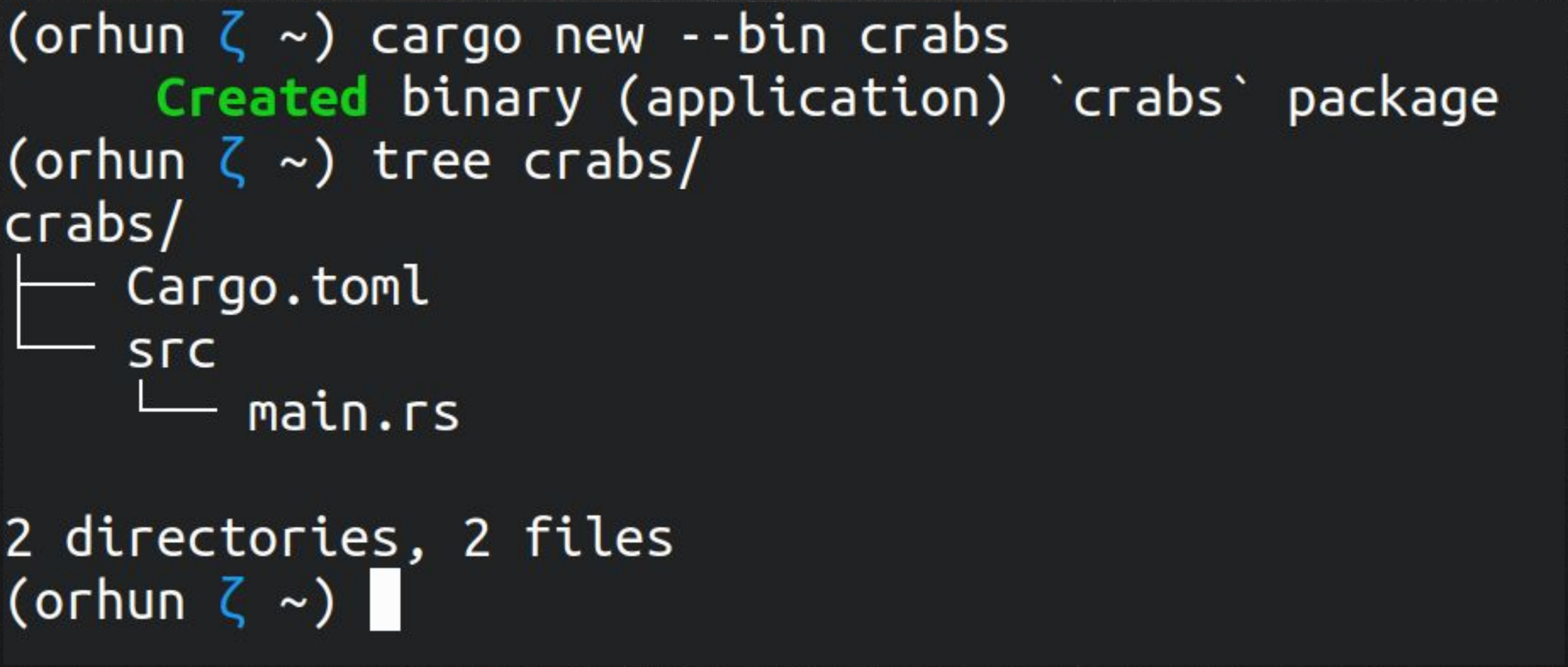
1. Create a new Rust (binary) package using Cargo and name it “crabs”.
2. Add “rand” crate (dependency) to the project.
3. “println!” a random number using `rand::random`



“Zero-dependency random number generation in Rust”

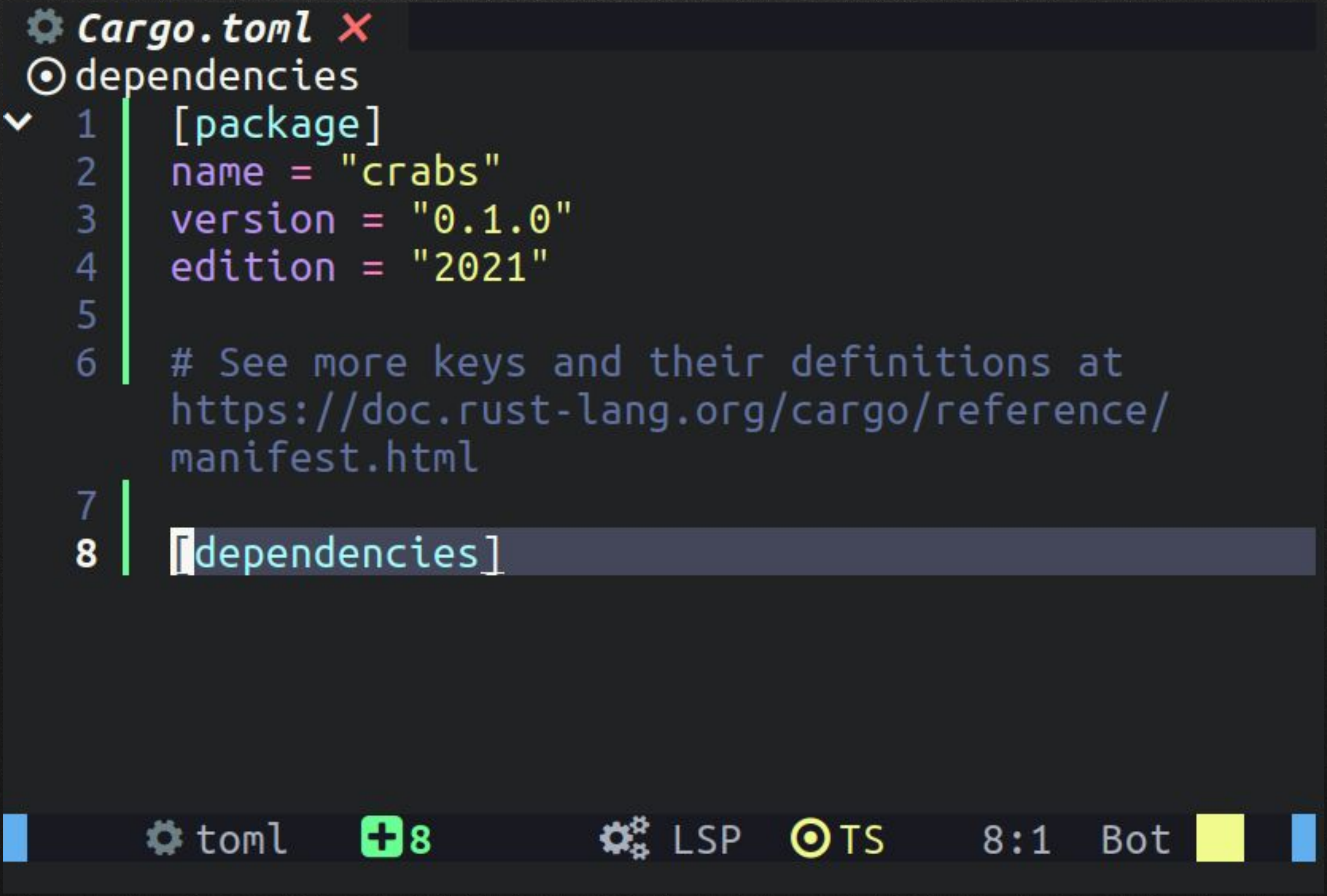
https://blog.orhun.dev/zero-deps-random-in-rust/

Creating a new Rust package



```
(orhun ζ ~) cargo new --bin crabs
     Created binary (application) `crabs` package
(orhun ζ ~) tree crabs/
crabs/
├── Cargo.toml
└── src
    └── main.rs

2 directories, 2 files
(orhun ζ ~) █
```

Inspecting Cargo.toml

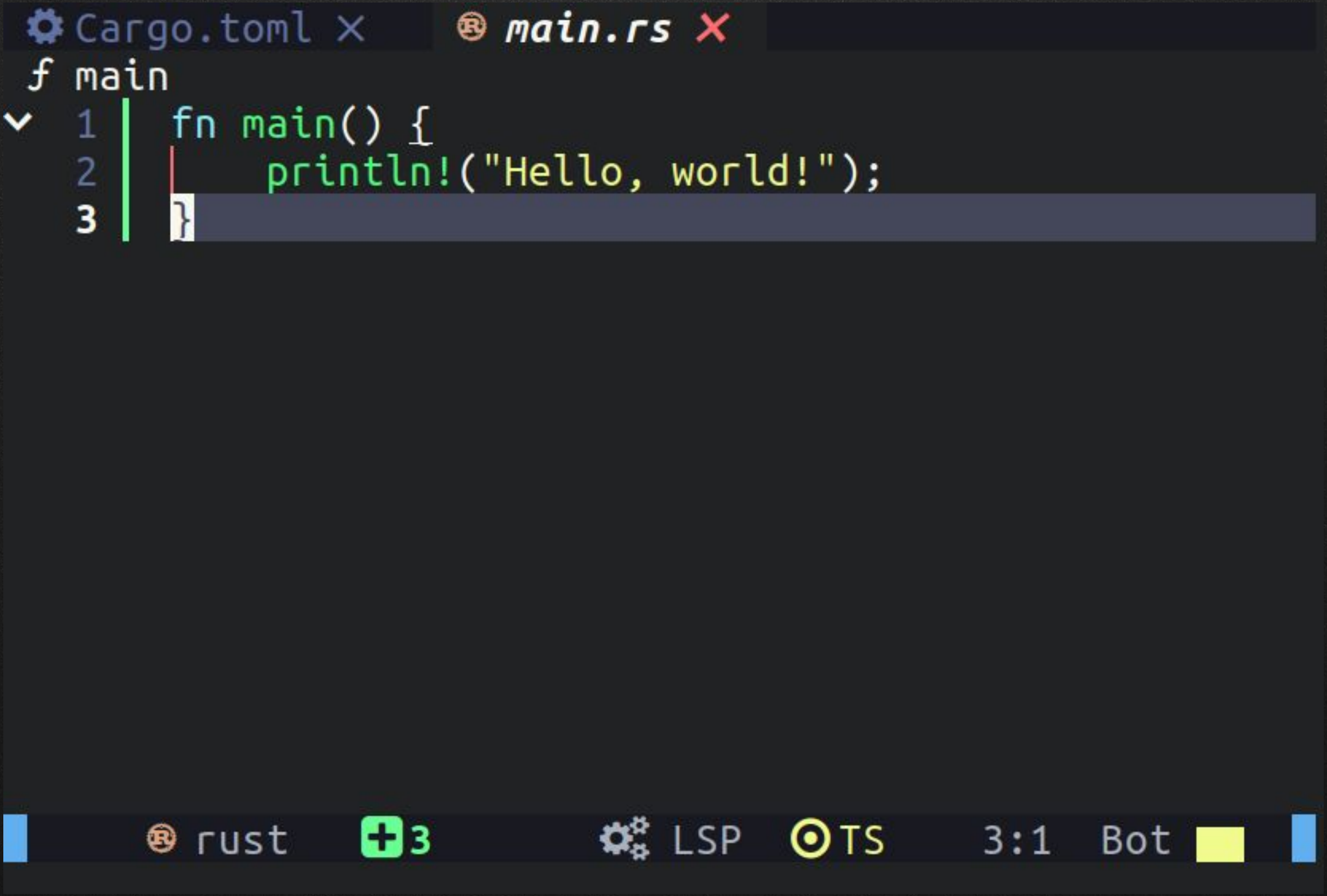


Cargo.toml

dependencies

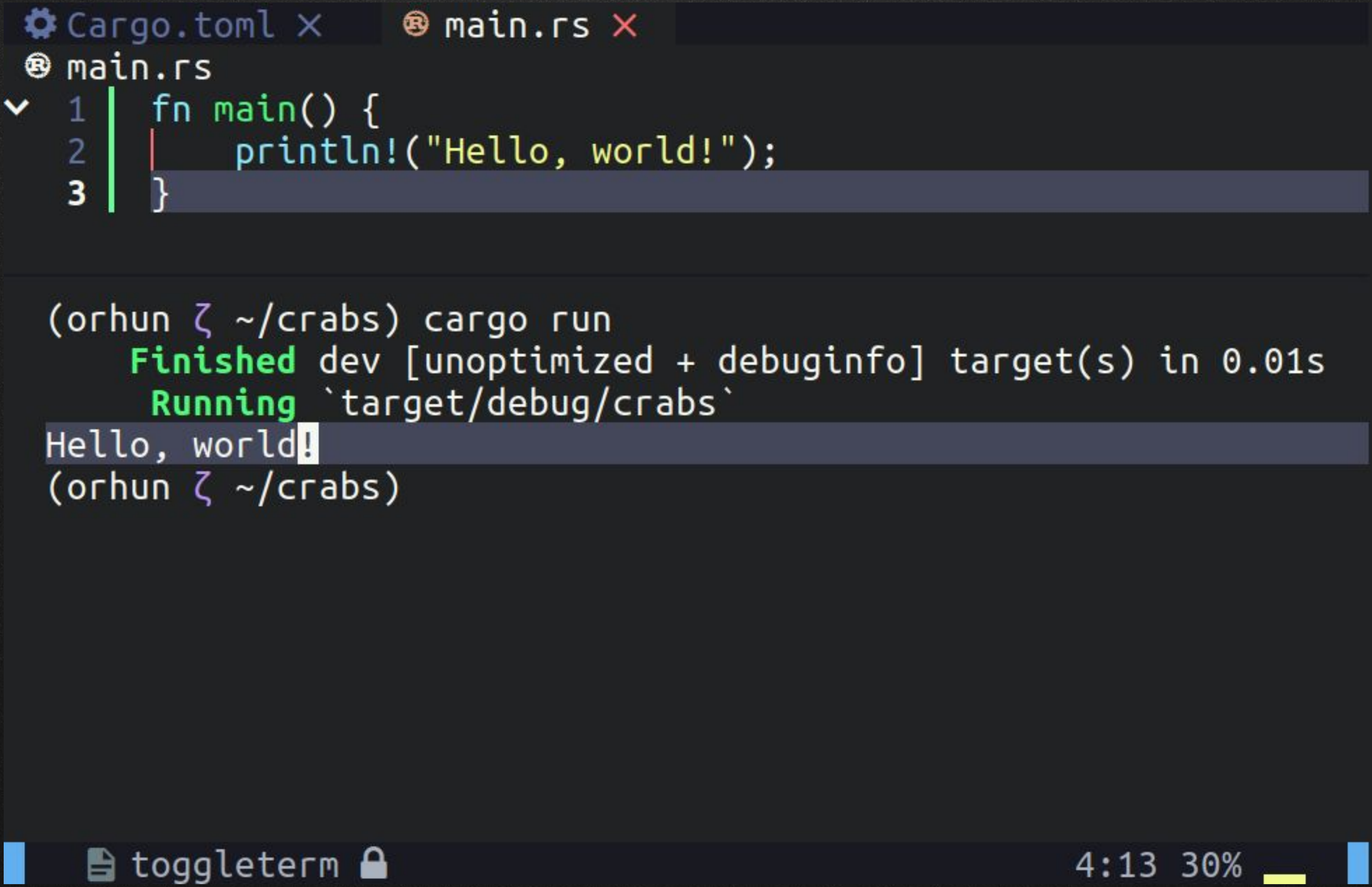
```toml
[package]
name = "crabs"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
```

Inspecting src/main.rs



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

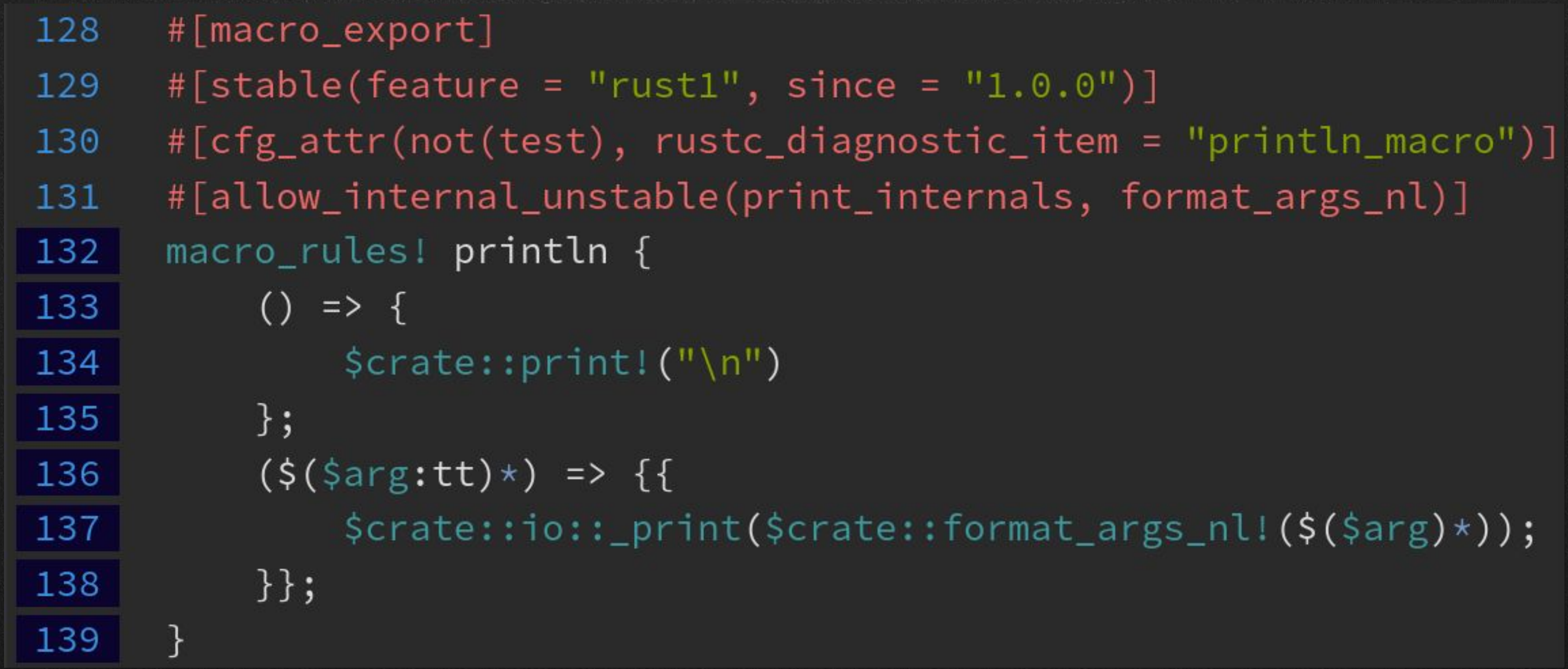
Running the project



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

```
(orhun ζ ~/crabs) cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
     Running `target/debug/crabs`
Hello, world!
(orhun ζ ~/crabs)
```

What is println!?



```
128  #[macro_export]
129  #[stable(feature = "rust1", since = "1.0.0")]
130  #[cfg_attr(not(test), rustc_diagnostic_item = "println_macro")]
131  #[allow_internal_unstable(print_internals, format_args_nl)]
132  macro_rules! println {
133      () => {
134          $crate::print!("\n")
135      };
136      ($($arg:tt)*) => {{
137          $crate::io::_print($crate::format_args_nl!($($arg)*));
138      }};
139  }
```

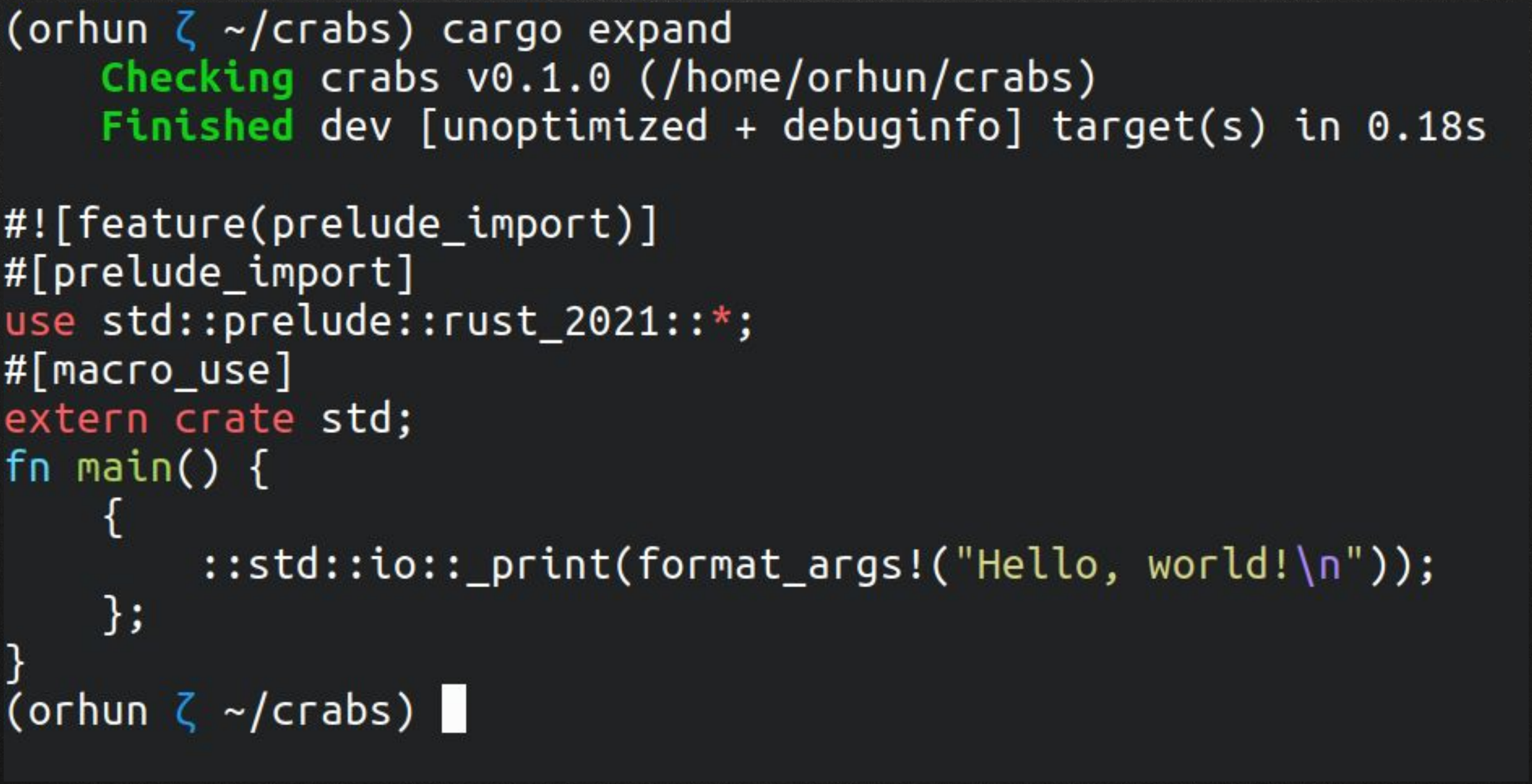
https://doc.rust-lang.org/src/std/macros.rs.html#132-139
https://doc.rust-lang.org/book/ch19-06-macros.html

cargo-expand

Subcommand to show result of macro expansion.

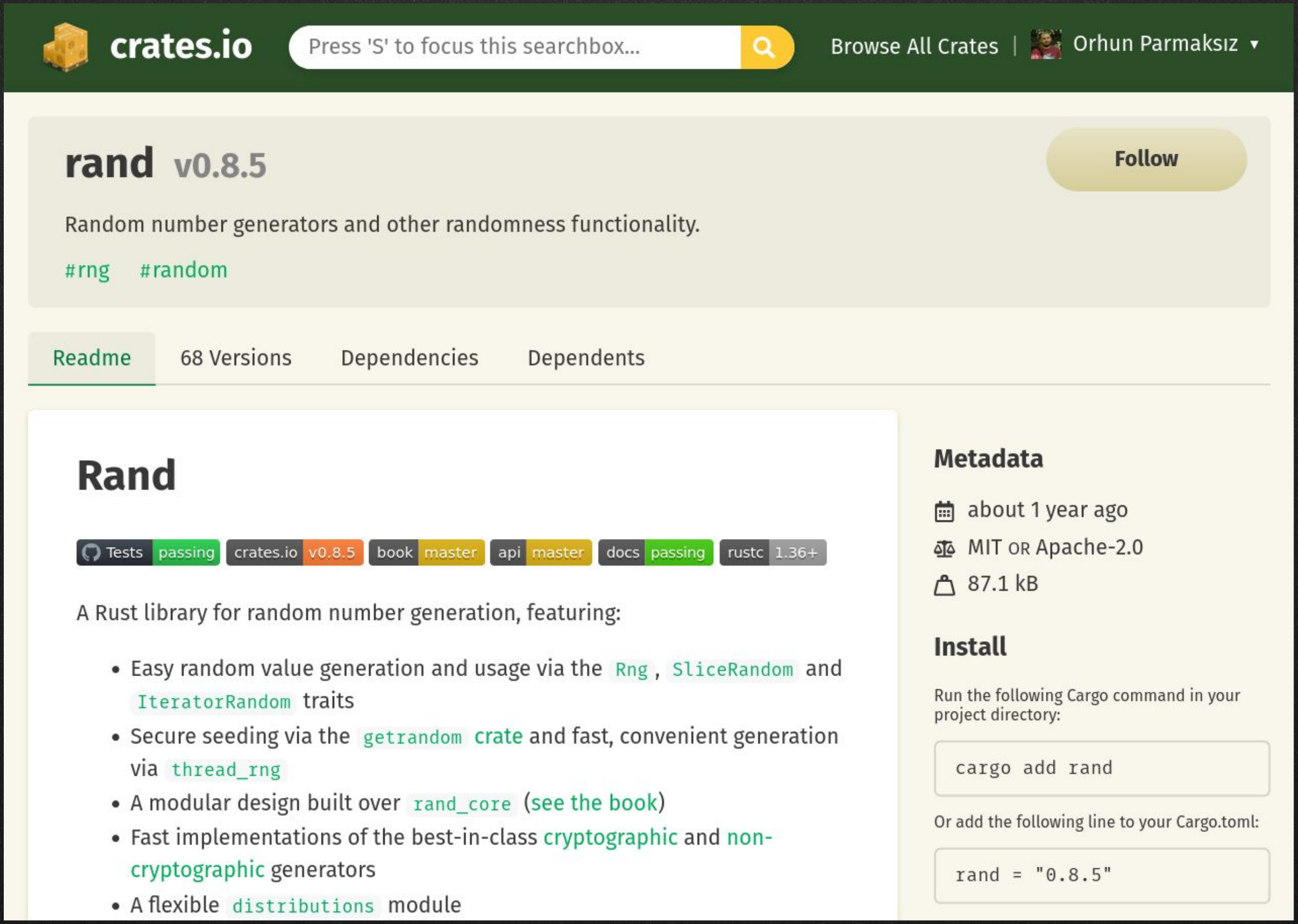
Install with `cargo install cargo-expand`

https://github.com/dtolnay/cargo-expand



```
(orhun ζ ~/crabs) cargo expand
    Checking crabs v0.1.0 (/home/orhun/crabs)
    Finished dev [unoptimized + debuginfo] target(s) in 0.18s

#![feature(prelude_import)]
#[prelude_import]
use std::prelude::rust_2021::*;
#[macro_use]
extern crate std;
fn main() {
    {
        ::std::io::_print(format_args!("Hello, world!\n"));
    };
}
(orhun ζ ~/crabs)
```

Let’s add rand dependency



rand v0.8.5

Random number generators and other randomness functionality.

#rng #random

Rand

A Rust library for random number generation, featuring:

- Easy random value generation and usage via the `Rng`, `SliceRandom` and `IteratorRandom` traits
- Secure seeding via the `getrandom` crate and fast, convenient generation via `thread_rng`
- A modular design built over `rand_core` (see the book)
- Fast implementations of the best-in-class cryptographic and non-cryptographic generators
- A flexible `distributions` module

Metadata

about 1 year ago
MIT OR Apache-2.0
87.1 kB

Install

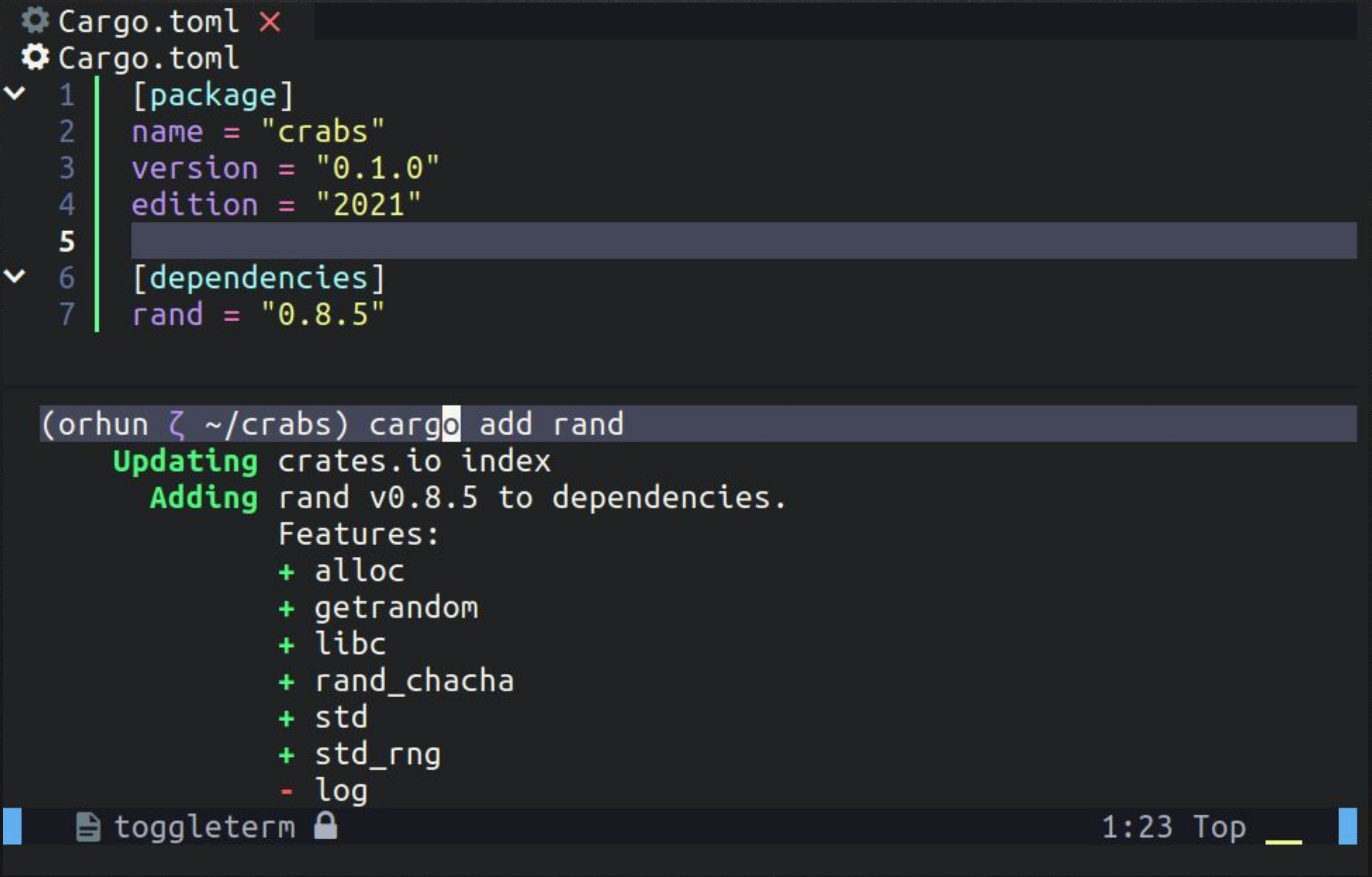
Run the following Cargo command in your project directory:

```
cargo add rand
```

Or add the following line to your Cargo.toml:

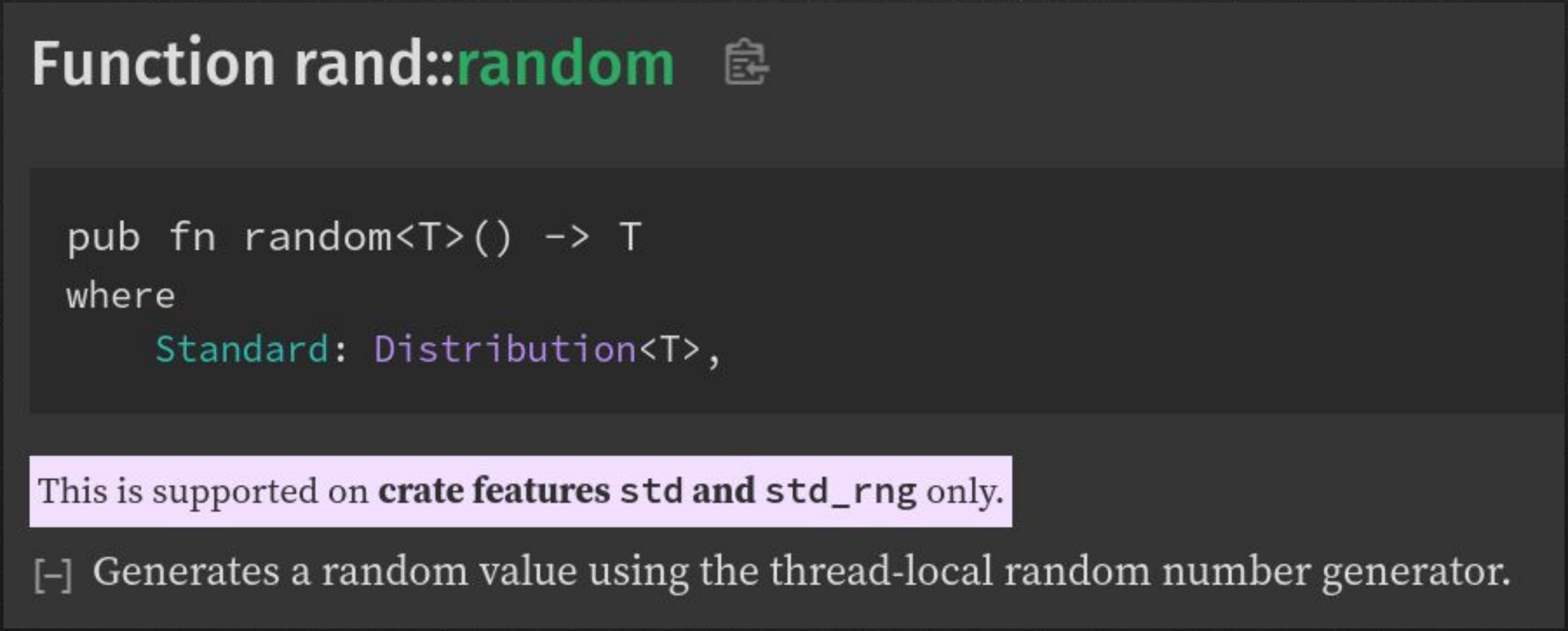
```
rand = "0.8.5"
```

https://crates.io/crates/rand



```
Cargo.toml ×
Cargo.toml
[package]
name = "crabs"
version = "0.1.0"
edition = "2021"

[dependencies]
rand = "0.8.5"
```

```
(orhun ζ ~/crabs) cargo add rand
    Updating crates.io index
      Adding rand v0.8.5 to dependencies.
             Features:
             + alloc
             + getrandom
             + libc
             + rand_chacha
             + std
             + std_rng
             - log
```



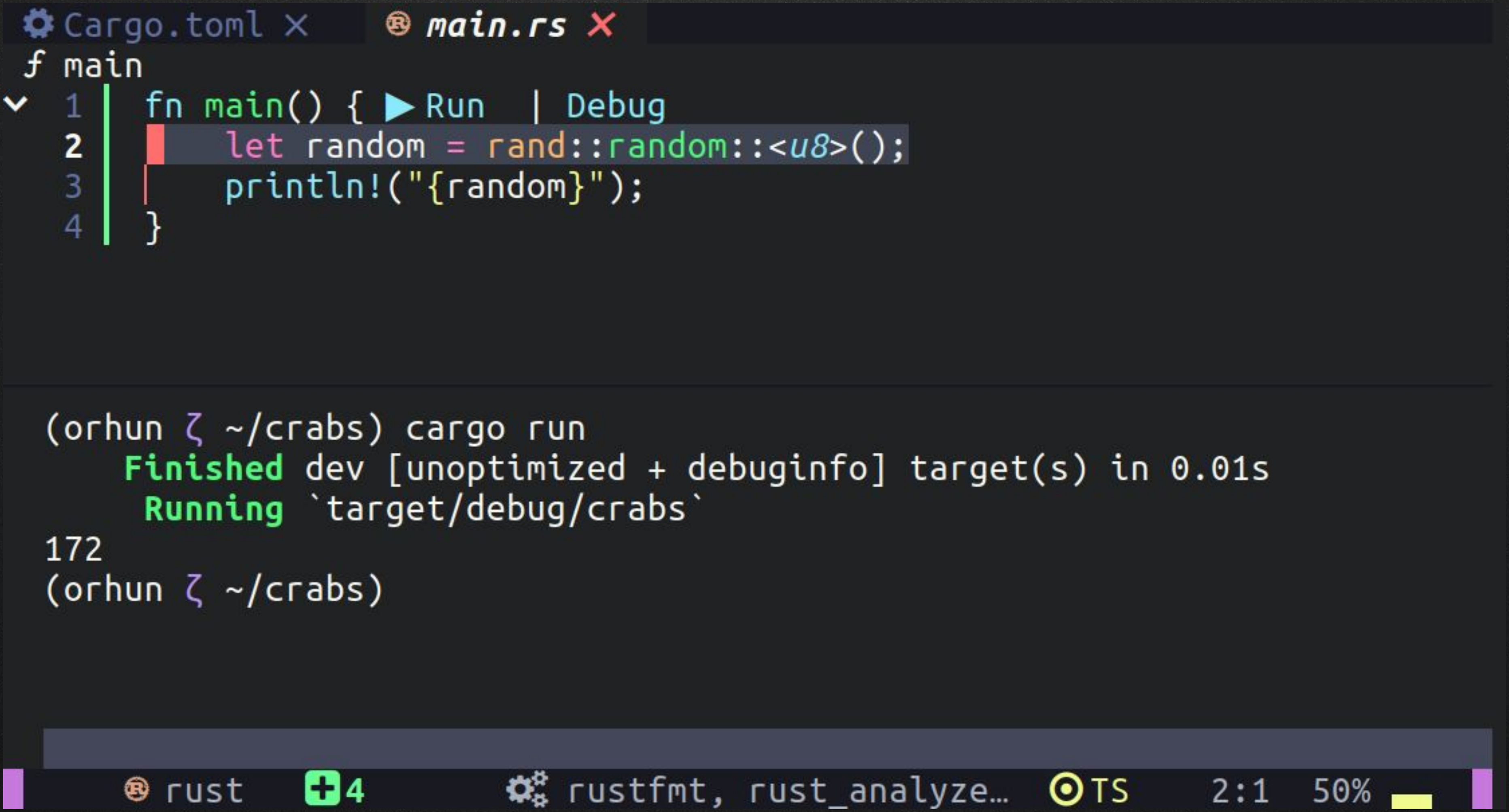
Function rand::random

```
pub fn random<T>() -> T
where
    Standard: Distribution<T>,
```

This is supported on **crate features `std` and `std_rng`** only.

[−] Generates a random value using the thread-local random number generator.

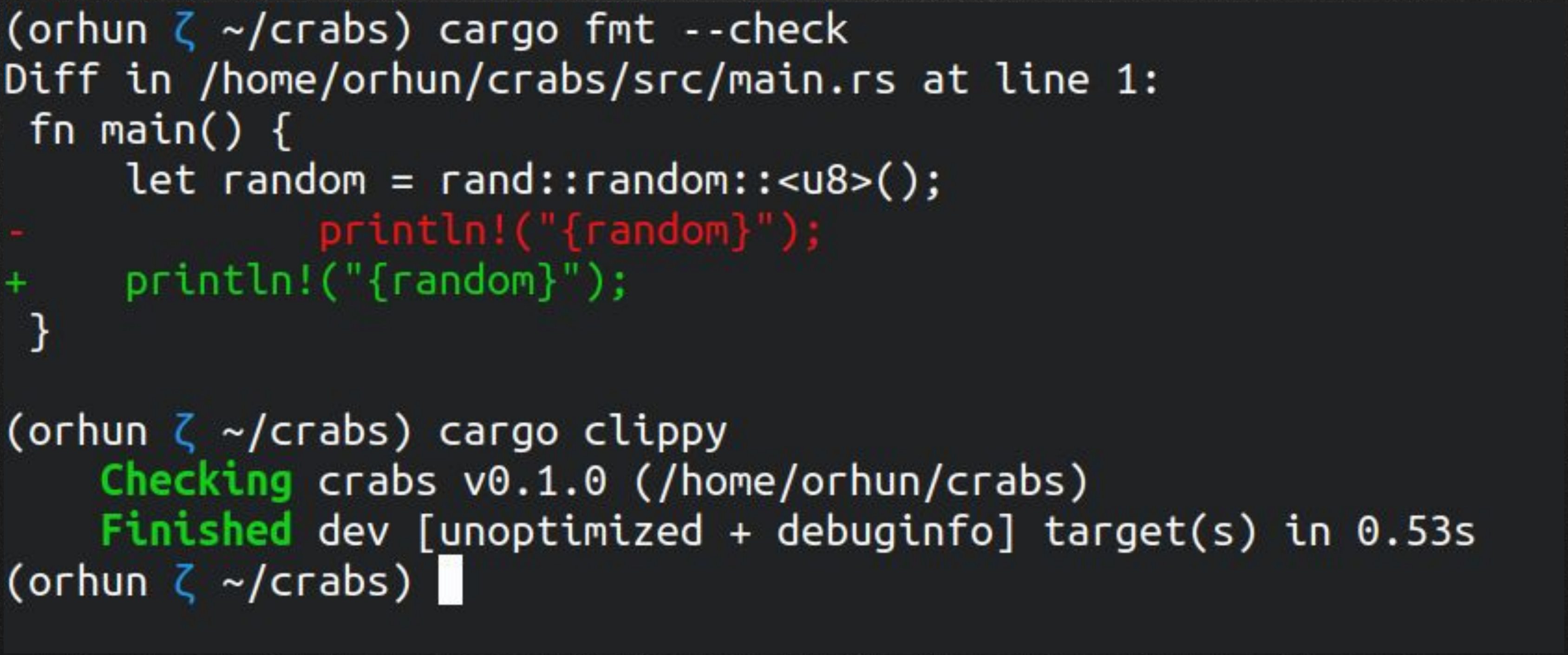
https://docs.rs/rand/latest/rand/fn.random.html



```rust
fn main() {
    let random = rand::random::<u8>();
    println!("{random}");
}
```

```
(orhun ζ ~/crabs) cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
     Running `target/debug/crabs`
172
(orhun ζ ~/crabs)
```

Check formatting / lints

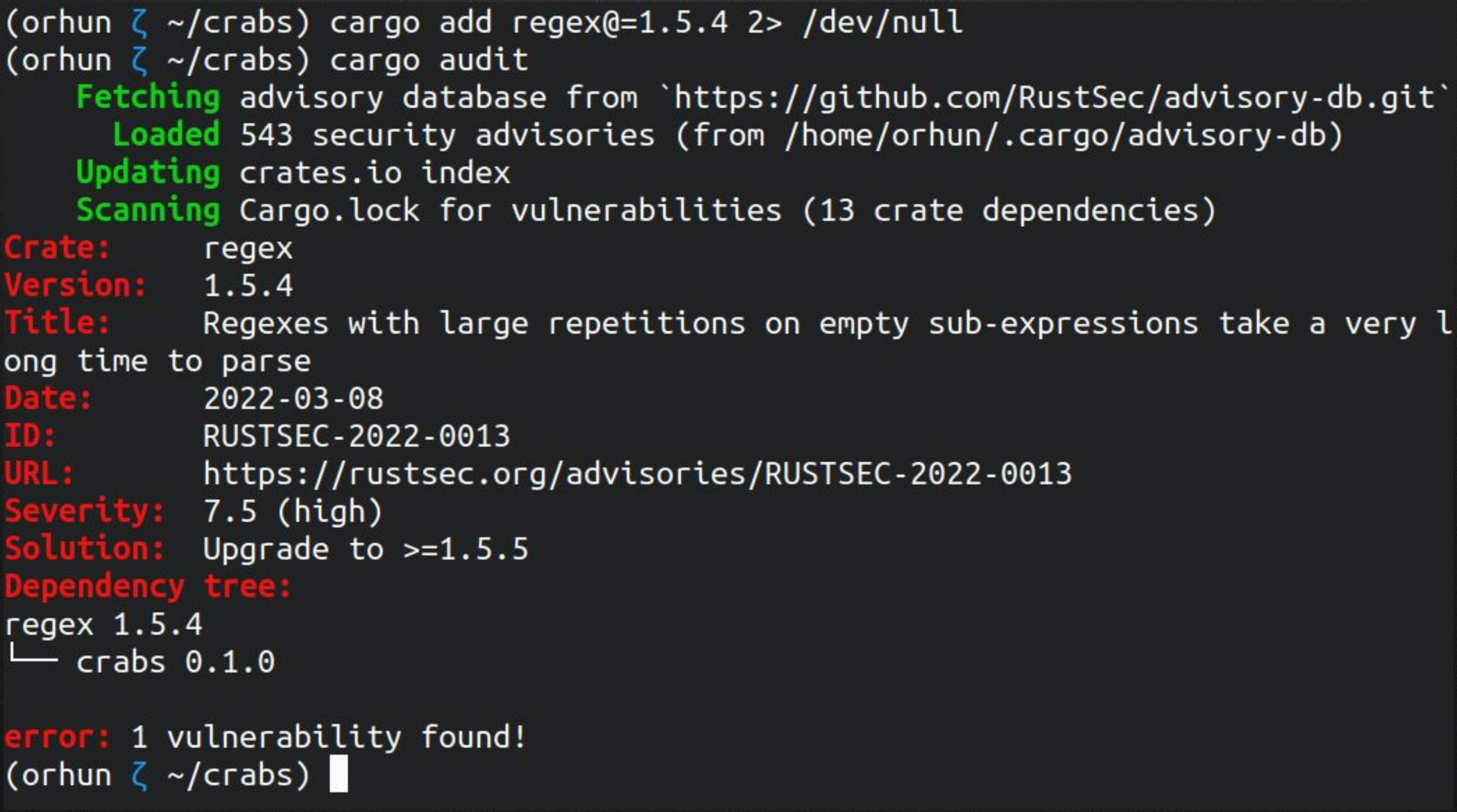


```
(orhun ζ ~/crabs) cargo fmt --check
Diff in /home/orhun/crabs/src/main.rs at line 1:
 fn main() {
     let random = rand::random::<u8>();
-            println!("{random}");
+    println!("{random}");
 }

(orhun ζ ~/crabs) cargo clippy
    Checking crabs v0.1.0 (/home/orhun/crabs)
    Finished dev [unoptimized + debuginfo] target(s) in 0.53s
(orhun ζ ~/crabs) 
```

Tools

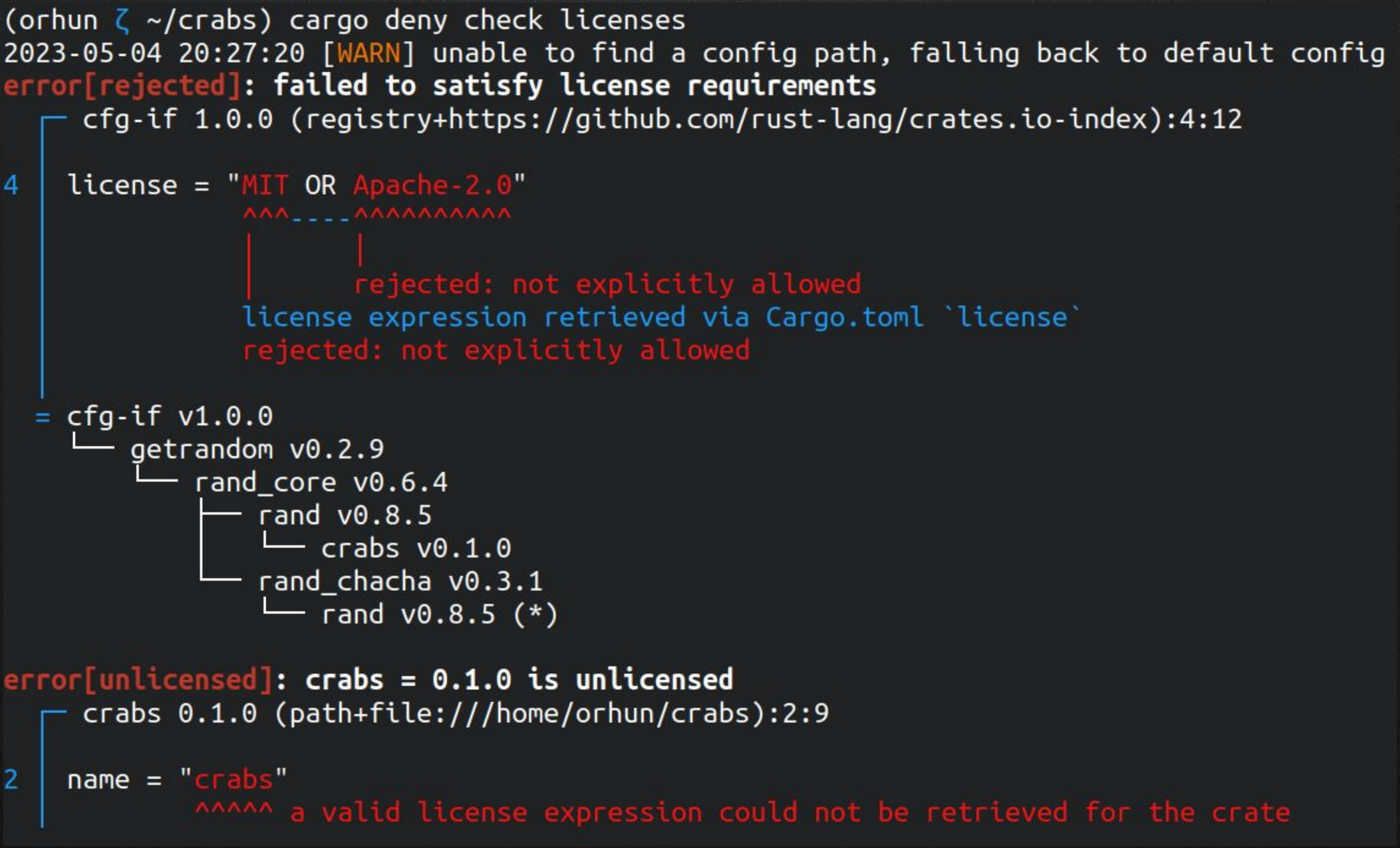
- cargo-audit
  - https://github.com/rustsec/rustsec
- cargo-deny
  - https://github.com/EmbarkStudios/cargo-deny
- cargo-msrv
  - https://github.com/foresterre/cargo-msrv
- cargo-spellcheck
  - https://github.com/drahnr/cargo-spellcheck
- cargo-outdated
  - https://github.com/kbknapp/cargo-outdated
- cargo-bloat
  - https://github.com/RazrFalcon/cargo-bloat
- cargo-udeps
  - https://github.com/est31/cargo-udeps

cargo-audit



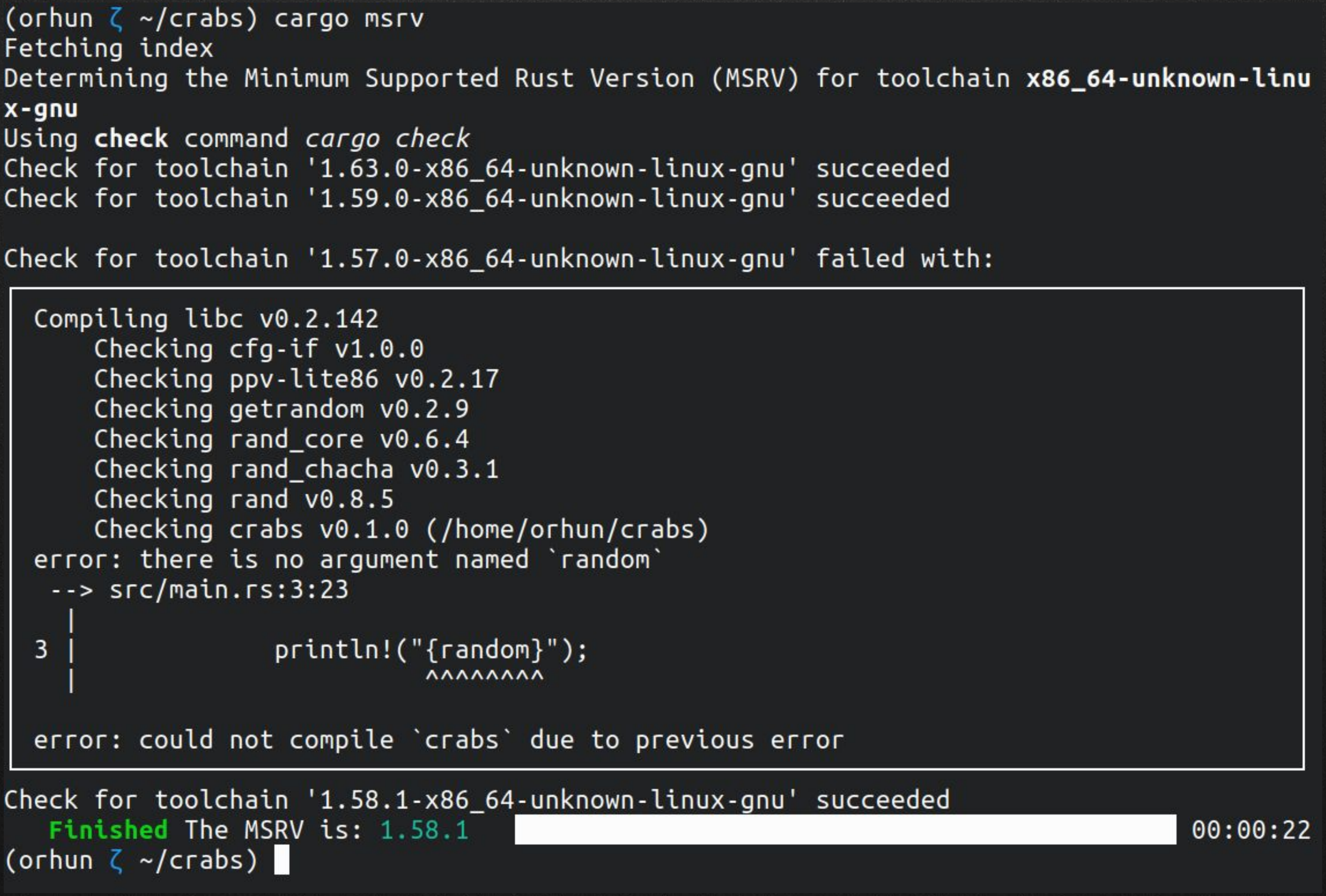
```
(orhun ζ ~/crabs) cargo add regex@=1.5.4 2> /dev/null
(orhun ζ ~/crabs) cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 543 security advisories (from /home/orhun/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (13 crate dependencies)
Crate:     regex
Version:   1.5.4
Title:     Regexes with large repetitions on empty sub-expressions take a very l
ong time to parse
Date:      2022-03-08
ID:        RUSTSEC-2022-0013
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0013
Severity:  7.5 (high)
Solution:  Upgrade to >=1.5.5
Dependency tree:
regex 1.5.4
└── crabs 0.1.0

error: 1 vulnerability found!
(orhun ζ ~/crabs)
```

cargo-deny



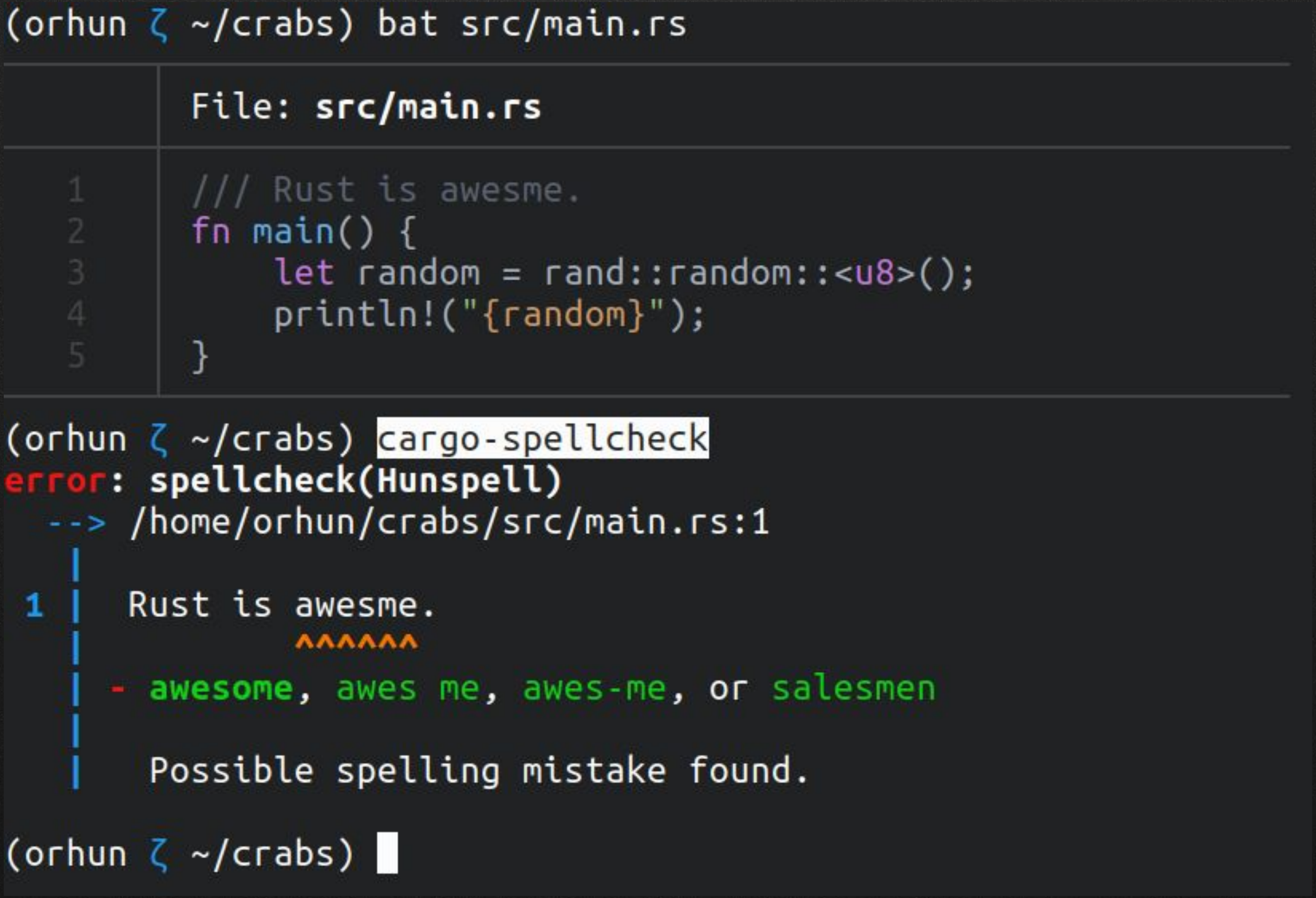
```
(orhun ζ ~/crabs) cargo deny check licenses
2023-05-04 20:27:20 [WARN] unable to find a config path, falling back to default config
error[rejected]: failed to satisfy license requirements
   ┌─ cfg-if 1.0.0 (registry+https://github.com/rust-lang/crates.io-index):4:12
   │
4  │ license = "MIT OR Apache-2.0"
   │            ^^^----^^^^^^^^^^
   │            │      │
   │            │      rejected: not explicitly allowed
   │            license expression retrieved via Cargo.toml `license`
   │            rejected: not explicitly allowed
   │
   = cfg-if v1.0.0
     └── getrandom v0.2.9
         └── rand_core v0.6.4
             ├── rand v0.8.5
             │   └── crabs v0.1.0
             └── rand_chacha v0.3.1
                 └── rand v0.8.5 (*)

error[unlicensed]: crabs = 0.1.0 is unlicensed
   ┌─ crabs 0.1.0 (path+file:///home/orhun/crabs):2:9
   │
2  │ name = "crabs"
   │         ^^^^^ a valid license expression could not be retrieved for the crate
```

cargo-msrv



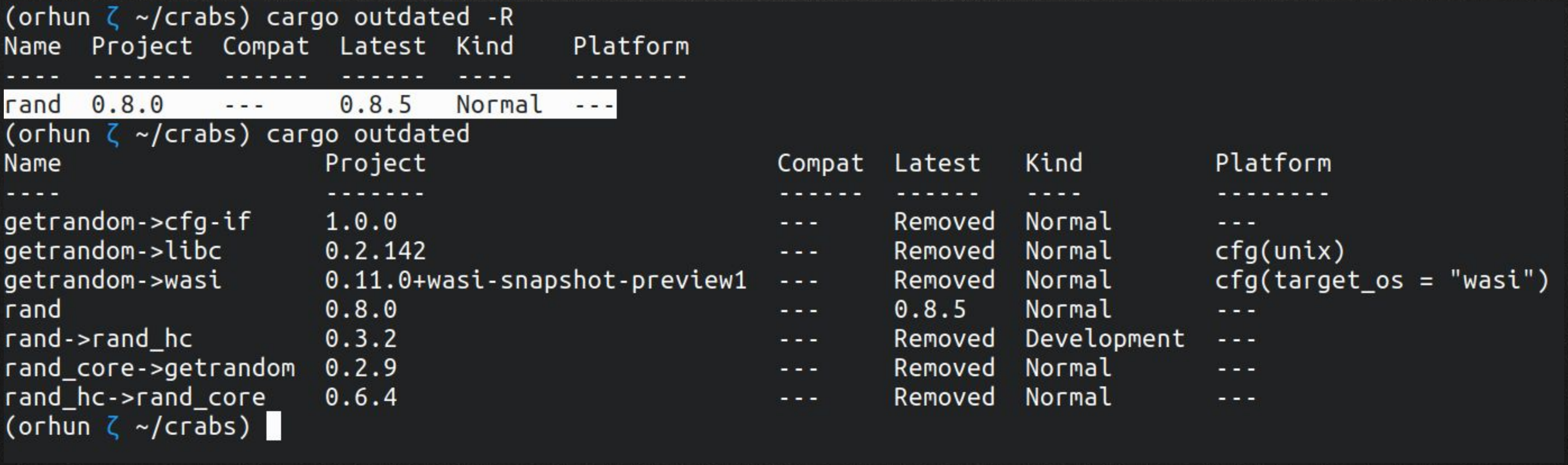
```
(orhun ζ ~/crabs) cargo msrv
Fetching index
Determining the Minimum Supported Rust Version (MSRV) for toolchain x86_64-unknown-linu
x-gnu
Using check command cargo check
Check for toolchain '1.63.0-x86_64-unknown-linux-gnu' succeeded
Check for toolchain '1.59.0-x86_64-unknown-linux-gnu' succeeded

Check for toolchain '1.57.0-x86_64-unknown-linux-gnu' failed with:

 Compiling libc v0.2.142
     Checking cfg-if v1.0.0
     Checking ppv-lite86 v0.2.17
     Checking getrandom v0.2.9
     Checking rand_core v0.6.4
     Checking rand_chacha v0.3.1
     Checking rand v0.8.5
     Checking crabs v0.1.0 (/home/orhun/crabs)
 error: there is no argument named `random`
  --> src/main.rs:3:23
   |
 3 |             println!("{random}");
   |                       ^^^^^^^^

 error: could not compile `crabs` due to previous error

Check for toolchain '1.58.1-x86_64-unknown-linux-gnu' succeeded
   Finished The MSRV is: 1.58.1                                          00:00:22
(orhun ζ ~/crabs)
```

cargo-spellcheck



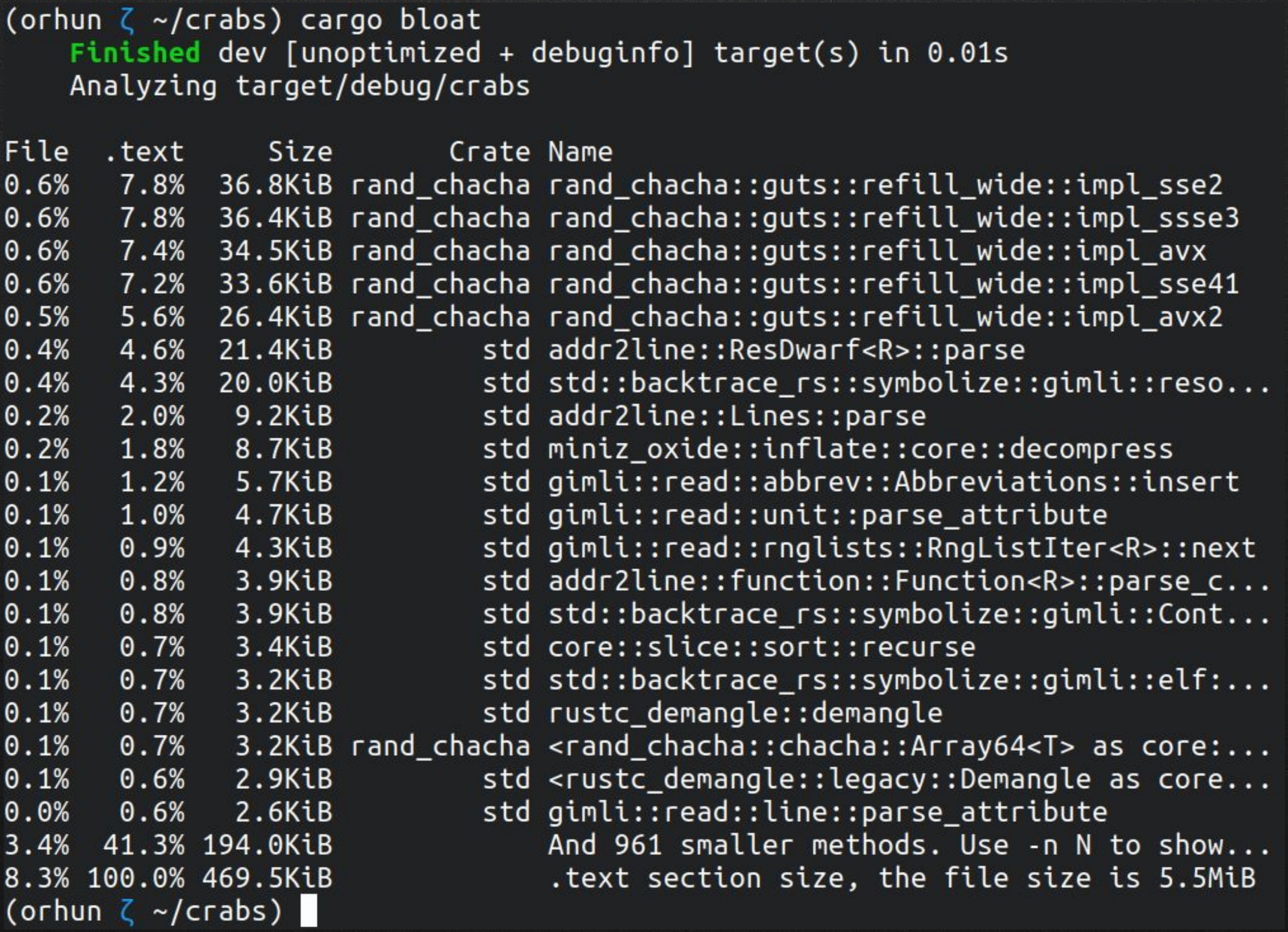
```
(orhun ζ ~/crabs) bat src/main.rs
───────┬────────────────────────────────────
       │ File: src/main.rs
───────┼────────────────────────────────────
       │ /// Rust is awesme.
       │ fn main() {
       │     let random = rand::random::<u8>();
       │     println!("{random}");
       │ }
───────┴────────────────────────────────────
(orhun ζ ~/crabs) cargo-spellcheck
error: spellcheck(Hunspell)
  --> /home/orhun/crabs/src/main.rs:1
   |
 1 |  Rust is awesme.
   |          ^^^^^^
   | - awesome, awes me, awes-me, or salesmen
   |
   |   Possible spelling mistake found.

(orhun ζ ~/crabs) █
```

cargo-outdated



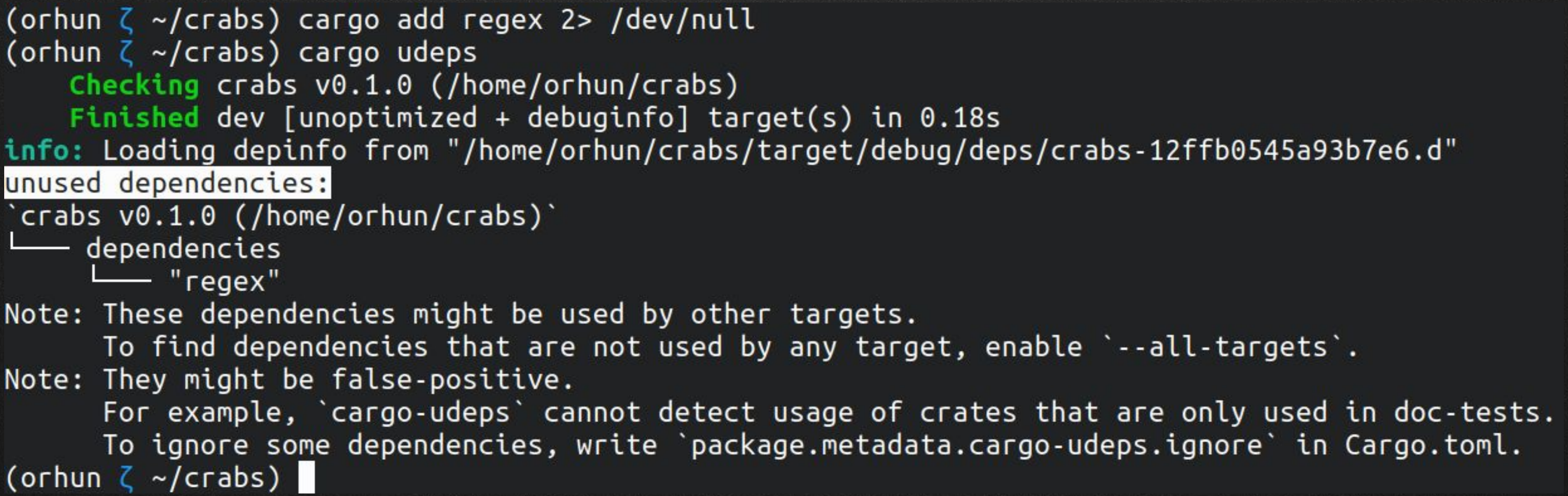
```
(orhun ζ ~/crabs) cargo outdated -R
Name  Project  Compat  Latest  Kind    Platform
----  -------  ------  ------  ----    --------
rand  0.8.0    ---     0.8.5   Normal  ---
(orhun ζ ~/crabs) cargo outdated
Name                    Project                              Compat  Latest   Kind         Platform
----                    -------                              ------  ------   ----         --------
getrandom->cfg-if       1.0.0                                ---     Removed  Normal       ---
getrandom->libc         0.2.142                              ---     Removed  Normal       cfg(unix)
getrandom->wasi         0.11.0+wasi-snapshot-preview1        ---     Removed  Normal       cfg(target_os = "wasi")
rand                    0.8.0                                ---     0.8.5    Normal       ---
rand->rand_hc           0.3.2                                ---     Removed  Development  ---
rand_core->getrandom    0.2.9                                ---     Removed  Normal       ---
rand_hc->rand_core      0.6.4                                ---     Removed  Normal       ---
(orhun ζ ~/crabs) 
```

cargo-bloat



```
(orhun ζ ~/crabs) cargo bloat
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
    Analyzing target/debug/crabs

File  .text     Size       Crate Name
0.6%   7.8%  36.8KiB rand_chacha rand_chacha::guts::refill_wide::impl_sse2
0.6%   7.8%  36.4KiB rand_chacha rand_chacha::guts::refill_wide::impl_ssse3
0.6%   7.4%  34.5KiB rand_chacha rand_chacha::guts::refill_wide::impl_avx
0.6%   7.2%  33.6KiB rand_chacha rand_chacha::guts::refill_wide::impl_sse41
0.5%   5.6%  26.4KiB rand_chacha rand_chacha::guts::refill_wide::impl_avx2
0.4%   4.6%  21.4KiB         std addr2line::ResDwarf<R>::parse
0.4%   4.3%  20.0KiB         std std::backtrace_rs::symbolize::gimli::reso...
0.2%   2.0%   9.2KiB         std addr2line::Lines::parse
0.2%   1.8%   8.7KiB         std miniz_oxide::inflate::core::decompress
0.1%   1.2%   5.7KiB         std gimli::read::abbrev::Abbreviations::insert
0.1%   1.0%   4.7KiB         std gimli::read::unit::parse_attribute
0.1%   0.9%   4.3KiB         std gimli::read::rnglists::RngListIter<R>::next
0.1%   0.8%   3.9KiB         std addr2line::function::Function<R>::parse_c...
0.1%   0.8%   3.9KiB         std std::backtrace_rs::symbolize::gimli::Cont...
0.1%   0.7%   3.4KiB         std core::slice::sort::recurse
0.1%   0.7%   3.2KiB         std std::backtrace_rs::symbolize::gimli::elf:...
0.1%   0.7%   3.2KiB         std rustc_demangle::demangle
0.1%   0.7%   3.2KiB rand_chacha <rand_chacha::chacha::Array64<T> as core:...
0.1%   0.6%   2.9KiB         std <rustc_demangle::legacy::Demangle as core...
0.0%   0.6%   2.6KiB         std gimli::read::line::parse_attribute
3.4%  41.3% 194.0KiB             And 961 smaller methods. Use -n N to show...
8.3% 100.0% 469.5KiB             .text section size, the file size is 5.5MiB
(orhun ζ ~/crabs) █
```

cargo-udeps



```
(orhun ζ ~/crabs) cargo add regex 2> /dev/null
(orhun ζ ~/crabs) cargo udeps
    Checking crabs v0.1.0 (/home/orhun/crabs)
    Finished dev [unoptimized + debuginfo] target(s) in 0.18s
info: Loading depinfo from "/home/orhun/crabs/target/debug/deps/crabs-12ffb0545a93b7e6.d"
unused dependencies:
`crabs v0.1.0 (/home/orhun/crabs)`
└─── dependencies
     └─── "regex"
Note: These dependencies might be used by other targets.
      To find dependencies that are not used by any target, enable `--all-targets`.
Note: They might be false-positive.
      For example, `cargo-udeps` cannot detect usage of crates that are only used in doc-tests.
      To ignore some dependencies, write `package.metadata.cargo-udeps.ignore` in Cargo.toml.
(orhun ζ ~/crabs) █
```

Summary

Rust is awesome.

Open source development with Rust is fun.

EOF

Thank you!

Any questions?

https://orhun.dev