Table of Contents

- 4.0

Rust

The Rust Programming Language
GitHub

nightly

- GitHub: https://github.com/KaiserY/rust-book-chinese
- GitBook: https://www.gitbook.com/book/kaisery/rust-book-chinese
- Rust : http://rust.cc/
- QQ : 144605258

- armink
- BingCheung
- Bluek404
- hczhcz
- honorabrutroll
- hy0kl
- JaySon-Huang
- KaiserY
- kenshinji
- kimw
- leqinotes
- linjx
- liubin
- liuzhe0223
- LuoZijun
- mapx
- NemoAlex
- peng1999
- quxiaolong1504
- t123yh
- ustcscgy
- ziqin
- 1989car

Rust

> README.md
> commit 3a6dbb30a21be8d237055479af613e30415b0c56

Rust Rust Rust“”Rust

Rust

- - Rust
- Rust - Rust
- - Rust
- Rust - Rust
- Rust -
- -
- - Rust Rust

“Rust”“”“ Rust”;“”

GitHub

> ## Rust
>
> Rust

Rust “”

```
fn main() {
    let mut x = vec!["Hello", "world"];
}
```

`x` `Vec<T>` vector `vec` `!` Rust `!`

`mut` `x` Rust vector

Rust Rust

Rust `x` `Vec<T>` vector “” Rust

“” Rust Rust `x` “” vector `x` vector Rust Rust `malloc` `free` ,

```
fn main() {
    let mut x = vec!["Hello", "world"];

    let y = &x[0];
}
```

`y` `y` vector “”Rust “”

```
fn main() {
    let mut x = vec!["Hello", "world"];

    let y = &x[0];

    x.push("foo");
}
```

`push` vector vector

```
error: cannot borrow `x` as mutable because it is also borrowed as immutable
    x.push(4);
    ^
note: previous borrow of `x` occurs here; the immutable borrow prevents
subsequent moves or mutable borrows of `x` until the borrow ends
    let y = &x[0];
             ^
note: previous borrow ends here
fn main() {

}
^
```

Rust `push` vector `y` vector 3 4 `y`
`y` “”(dangling pointer) `y`

```
fn main() {
    let mut x = vec!["Hello", "world"];

    let y = x[0].clone();

    x.push("foo");
}
```

Rust `clone()` `y` `x` vector `"hello"` `push()`

```
fn main() {
    let mut x = vec!["Hello", "world"];

    {
        let y = &x[0];
    }

    x.push("foo");
}
```

`y` `push()`

> getting-started.md
> commit 0b8370c3978bb47de97ce754ea601fc1b654cd2b

Rust Rust “Hello World”CargoRust

Rust

Rust Rust

`$` `$` `$` `#`

Rust Rust

“”"target triple"

T1 Tier 1

“”

-
- `rust-lang/rust` master
-
-

Target	std	rustc	cargo	notes
`x86_64-pc-windows-msvc`	✓	✓	✓	64-bit MSVC (Windows 7+)
`i686-pc-windows-gnu`	✓	✓	✓	32-bit MinGW (Windows 7+)
`x86_64-pc-windows-gnu`	✓	✓	✓	64-bit MinGW (Windows 7+)
`i686-apple-darwin`	✓	✓	✓	32-bit OSX (10.7+, Lion+)
`x86_64-apple-darwin`	✓	✓	✓	64-bit OSX (10.7+, Lion+)
`i686-unknown-linux-gnu`	✓	✓	✓	32-bit Linux (2.6.18+)
`x86_64-unknown-linux-gnu`	✓	✓	✓	64-bit Linux (2.6.18+)

T2 Tier 2

“”

-
- `rust-lang/rust` master   bootstrap
-

Target	std	rustc	cargo	notes
`i686-pc-windows-msvc`	✓	✓	✓	32-bit MSVC (Windows 7+)
`x86_64-unknown-linux-musl`	✓			64-bit Linux with MUSL
`arm-linux-androideabi`	✓			ARM Android
`arm-unknown-linux-gnueabi`	✓	✓		ARM Linux (2.6.18+)
`arm-unknown-linux-gnueabihf`	✓	✓		ARM Linux (2.6.18+)
`aarch64-unknown-linux-gnu`	✓			ARM64 Linux (2.6.18+)
`mips-unknown-linux-gnu`	✓			MIPS Linux (2.6.18+)
`mipsel-unknown-linux-gnu`	✓			MIPS (LE) Linux (2.6.18+)

T3 Tier 3Tengu

Rust bug

Target	std	rustc	cargo	notes
`i686-linux-android`	✓			32-bit x86 Android
`aarch64-linux-android`	✓			ARM64 Android
`powerpc-unknown-linux-gnu`	✓			PowerPC Linux (2.6.18+)
`i386-apple-ios`	✓			32-bit x86 iOS
`x86_64-apple-ios`	✓			64-bit x86 iOS
`armv7-apple-ios`	✓			ARM iOS
`armv7s-apple-ios`	✓			ARM iOS
`aarch64-apple-ios`	✓			ARM64 iOS
`i686-unknown-freebsd`	✓	✓		32-bit FreeBSD
`x86_64-unknown-freebsd`	✓	✓		64-bit FreeBSD
`x86_64-unknown-openbsd`	✓	✓		64-bit OpenBSD
`x86_64-unknown-netbsd`	✓	✓		64-bit NetBSD
`x86_64-unknown-bitrig`	✓	✓		64-bit Bitrig
`x86_64-unknown-dragonfly`	✓	✓		64-bit DragonFlyBSD
`x86_64-rumprun-netbsd`	✓			64-bit NetBSD Rump Kernel
`i686-pc-windows-msvc` (XP)	✓			Windows XP support
`x86_64-pc-windows-msvc` (XP)	✓			Windows XP support

Linux Mac

Linux Mac

```
$ curl -sSf https://static.rust-lang.org/rustup.sh | sh
```

```
Welcome to Rust.

This script will download the Rust compiler and its package manager, Cargo, and
install them to /usr/local. You may install elsewhere by running this script
with the --prefix=<path> option.

The installer will run under ‘sudo’ and may ask you for your password. If you do
not want the script to run ‘sudo’ then pass it the --disable-sudo flag.

You may uninstall later by running /usr/local/lib/rustlib/uninstall.sh,
or by running this script again with the --uninstall flag.

Continue? (y/N)
```

`y` `yes`

Windows

Windows

Rust Linux Mac

```
$ sudo /usr/local/lib/rustlib/uninstall.sh
```

Windows `.msi`

Troubleshooting

Rust shell

```
$ rustc --version
```

hash

Rust

Windows Rust `%PATH%` “Change, repair, or remove installation”“Change”“Add to PATH”

Mibbit irc.mozilla.org #rust IRC RustaceansRuster Stack Overflow

UNIX `/usr/local/share/doc/rust` Windows Rust `share/doc`

Hello, world!

Rust Rust “Hello, world!”

> Rust IDE SolidOak Rust Rust IDE Rust IDE

Rust Rust home

```
$ mkdir ~/projects
$ cd ~/projects
$ mkdir hello_world
$ cd hello_world
```

> Windows PowerShell `~` Shell

Rust

`main.rs` Rust `.rs` Rust `hello_world.rs` `helloworld.rs`

`main.rs`

+code%20%7B%0A%20%20%20%20println!(%22Hello%2C%20world!%22)%3B%0A%7D%0A)

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

Linux OSX

```
$ rustc main.rs
$ ./main
Hello, world!
```

Windows `main.exe` `main` `Hello, world!` Rust Rust

Rust

“Hello, world!”

+code%20%7B%0A%0A%7D%0A)

```rust
fn main() {

}
```

Rust `main` Rust “ `main` ” `( )`

`{ }` Rust

`main()`

+code%20%7B%0A%20%20%20%20%20%20%20%20println!(%22Hello%2C%20world!%22)%3B%0A%20%20%20%20%0A%7D)

```rust
println!("Hello, world!");
```

4

`println!()` Rust Rust `println()` ! Rust `!`

`"Hello, world!"` `println!` (⊙o⊙

`;` Rust `;` Rust `;`

“ Rust ”

Rust `rustc` Rust

```
$ rustc main.rs
```

C C++ `gcc` `clang` Rust Linux OSX shell `ls`

```
$ ls
main  main.rs
```

Windows

```
$ dir
main.exe  main.rs
```

`.rs` Windows `main.exe` `main` `main` `main.exe`

```
$ ./main  # or main.exe on Windows
```

`main.rs` “Hello, world!” `Hello, world!`

RubyPython JavaScript Rust *ahead-of-time compiled language* Rust `.rb` `.py` `.js` RubyPythonJavaScript `rustc` Cargo Rust

Hello, Cargo!

Cargo Rust Rustacean Cargo Rust Cargo “dependencies”

Rust Rust Cargo

Rust CargoRust Cargo Rust Cargo

```
$ cargo --version
```

OK“ `command not found` ” Rust Cargo

Cargo

Hello World Cargo Cargo

1.
2. Windows `main.exe` `main`
3. Cargo

`hello_world`

```
$ mkdir src
$ mv main.rs src/main.rs
$ rm main  # or 'del main.exe' on Windows
```

Cargo src hello_world READMElicense Cargo

`main.rs` `src` `rustc` Windows `main.exe` `main`

`main.rs` `lib.rs` Cargo

`hello_world` `Cargo.toml`

`Cargo.toml` `C` Cargo

TOMLTom's Obvious, Minimal Language TOML INI Cargo

```
[package]

name = "hello_world"
version = "0.0.1"
authors = [ "Your name <you@example.com>" ]
```

`[package]`

Cargo

`Cargo.toml`

Cargo

`Cargo.toml` Hello World

```
$ cargo build
   Compiling hello_world v0.0.1 (file:///home/yourname/projects/hello_world)
$ ./target/debug/hello_world
Hello, world!
```

`Hello, world!`

`cargo build` `./target/debug/hello_world` `cargo run`

```
$ cargo run
     Running `target/debug/hello_world`
Hello, world!
```

Cargo Cargo

```
$ cargo run
   Compiling hello_world v0.0.1 (file:///home/yourname/projects/hello_world)
     Running `target/debug/hello_world`
Hello, world!
```

Cargo

Cargo `rustc` crate Cargo Cargo `cargo build`

Building for Release

`cargo build --release` Rust

Cargo `Cargo.lock`

```
[root]
name = "hello_world"
version = "0.0.1"
```

Cargo `Cargo.lock` Hello World `Cargo.lock` Cargo

Cargo `hello_world`

Rust Rust

```
$ git clone someurl.com/foo
$ cd foo
$ cargo build
```

Cargo

Cargo

Cargo `cargo new`

```
$ cargo new hello_world --bin
```

`--bin` `/usr/bin` Unix

Cargo `Cargo.toml` `main.rs` `src`

`Cargo.toml`

```
[package]

name = "hello_world"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]
```

Cargo `git` Cargo `hello_world` `git`

`src/main.rs`

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

Cargo “Hello World”

> Cargo Cargo

Closing Thoughts

Rust Rust

“ Rust ” “ ” “ Rust”

> guessing-game.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust 1 100

`hello_world` `Cargo.toml` Cargo

```
$ cd ~/projects
$ cargo new guessing_game --bin
$ cd guessing_game
```

`cargo new` `--bin`

`Cargo.toml`

```
[package]

name = "guessing_game"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]
```

Cargo

Cargo “Hello, world!” `src/main.rs`

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

Cargo

```
$ cargo build
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
```

`src/main.rs`

Cargo `run` `cargo run` `cargo build`

```
$ cargo run
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
     Running `target/debug/guessing_game`
Hello, world!
```

`run`

`src/main.rs`

```
use std::io;

fn main() {
    println!("Guess the number!");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guessed: {}", guess);
}
```

```
use std::io;
```

`io` Rust ‘prelude’ `use` "prelude", `io` prelude IO

```
fn main() {
```

`main()` `fn` `()` `{` `()`

```
    println!("Guess the number!");

    println!("Please input your guess.");
```

`println!()`

```
    let mut guess = String::new();
```

let“”

```
let foo = bar;
```

`foo` `bar` “" Rust

`mut` `let` pattern

```
let foo = 5; // immutable.
let mut bar = 5; // mutable
```

`//` Rust

`let mut guess` `guess` `=` `String::new()`

`String` StringUTF-8

`::new()` `::` ”“ `String` `String` “”

`new()` `String` `new()`

```
    io::stdin().read_line(&mut guess)
        .expect("Failed to read line");
```

```
io::stdin()
```

`use std::io` `use std::io` `std::io::stdin()`

std::io::Stdin

```
.read_line(&mut guess)
```

read_line() “” `read_line()` `&mut guess`

`guess` `read_line` `String` `&mut String` Rust“” Rust
`let` `&mut guess` `&guess`

`read_line()`

```
        .expect("Failed to read line");
```

`.foo()`

```
    io::stdin().read_line(&mut guess).expect("failed to read line");
```

3 3 `read_line()` `expect()` `read_line()` `&mut String` io::Result
Rust `Result` Result `io::Result`

`Result` `Result` `io::Result` expect() panic! `panic!`

```
$ cargo build
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
src/main.rs:10:5: 10:39 warning: unused result which must be used,
#[warn(unused_must_use)] on by default
src/main.rs:10     io::stdin().read_line(&mut guess);
                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Rust `Result` `io::Result` Rust

```
    println!("You guessed: {}", guess);
}
```

`{}` `guess` `{}`

```
let x = 5;
let y = 10;

println!("x and y: {} and {}", x, y);
```

`cargo run`

```
$ cargo run
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
     Running `target/debug/guessing_game`
Guess the number!
Please input your guess.
6
You guessed: 6
```

RustRust `rand` crate“”crate Rust ”“ `rand` ”“

Cargo `rand` `Cargo.toml`

```
[dependencies]

rand="0.3.0"
```

`Cargo.toml` `[dependencies]` `[package]` Cargo `0.3.0` Cargo `0.3.0` `=0.3.0` `*` Cargo

```
$ cargo build
    Updating registry `https://github.com/rust-lang/crates.io-index`
 Downloading rand v0.3.8
 Downloading libc v0.1.6
   Compiling libc v0.1.6
   Compiling rand v0.3.8
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
```

Cargo Crates.ioCrates.io Rust Rust

Cargo `[dependencies]` `rand` `libc` `rand` `libc`

`cargo build`

```
$ cargo build
```

Cargo `src/main.rs`

```
$ cargo build
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
```

Cargo `0.3.x` `rand` `v0.3.8` `v0.3.9` bug bug `0.3.9`

`Cargo.lock` Cargo `Cargo.lock` Cargo `Cargo.lock` `0.3.8`

`v0.3.9` Cargo `update` “”Cargo `0.3.0` `0.4.0` `0.4.x` `Cargo.toml` `cargo build` Cargo `rand`

CargoCargoRustacean

`rand`

```rust
extern crate rand;

use std::io;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("failed to read line");

    println!("You guessed: {}", guess);
}
```

`extern crate rand` `[dependencies]` `rand` `extern crate` Rust `use rand;` `rand::` `rand`

`use` `use rand::Rng` `Rng` “traits” trait

```rust
    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);
```

`rand::thread_rng()` `use rand::Rng` `gen_range()` `1` `101` `1` `100`

```
$ cargo run
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
     Running `target/debug/guessing_game`
Guess the number!
The secret number is: 7
Please input your guess.
4
You guessed: 4
$ cargo run
     Running `target/debug/guessing_game`
Guess the number!
The secret number is: 83
Please input your guess.
5
You guessed: 5
```

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("failed to read line");

    println!("You guessed: {}", guess);

    match guess.cmp(&secret_number) {
        Ordering::Less    => println!("Too small!"),
        Ordering::Greater => println!("Too big!"),
        Ordering::Equal   => println!("You win!"),
    }
}
```

`use std::cmp::Ordering` 5

```
match guess.cmp(&secret_number) {
    Ordering::Less    => println!("Too small!"),
    Ordering::Greater => println!("Too big!"),
    Ordering::Equal   => println!("You win!"),
}
```

`cmp()` `use Ordering` match `Ordering` `Ordering` enum

```
enum Foo {
    Bar,
    Baz,
}
```

`Foo` `Foo::Bar` `Foo::Baz` `::` `enum`

Ordering3 `Less` `Equal` `Greater` `match` “” 3 `Ordering` 3

```
match guess.cmp(&secret_number) {
    Ordering::Less    => println!("Too small!"),
    Ordering::Greater => println!("Too big!"),
    Ordering::Equal   => println!("You win!"),
}
```

`Less` `Too small!` `Greater` `Too big!` `Equal` `You win!` `match` Rust

```
$ cargo build
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
src/main.rs:28:21: 28:35 error: mismatched types:
 expected `&collections::string::String`,
    found `&_`
(expected struct `collections::string::String`,
    found integral variable) [E0308]
src/main.rs:28     match guess.cmp(&secret_number) {
                                   ^~~~~~~~~~~~~~
error: aborting due to previous error
Could not compile `guessing_game`.
```

“”Rust `let guess = String::new()` Rust `guess` `String` `secret_number` `1` `100` `i32` 32 `u32` 32 `i64` 64 Rust `i32` Rust `guess` `secret_number` `String` 3

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("failed to read line");

    let guess: u32 = guess.trim().parse()
        .expect("Please type a number!");

    println!("You guessed: {}", guess);

    match guess.cmp(&secret_number) {
        Ordering::Less    => println!("Too small!"),
        Ordering::Greater => println!("Too big!"),
        Ordering::Equal   => println!("You win!"),
    }
}
```

3

```
    let guess: u32 = guess.trim().parse()
        .expect("Please type a number!");
```

`guess` Rust “shadow” `guess` `guess` `String` `u32` Shadowing `guess` `guess_str` `guess`

`guess`

```
guess.trim().parse()
```

`guess` `guess` `String` `String` `trim()` “” `read_line()` `5` `guess` `5\n` `\n` “” `trim()` `5` `parse()` Rust `let guess: u32` `guess` `:` Rust `u32` 32Rust `u32`

`read_line()` `parse()` `A%?` `read_line()` `expect()`

```
$ cargo run
   Compiling guessing_game v0.0.1 (file:///home/you/projects/guessing_game)
     Running `target/guessing_game`
Guess the number!
The secret number is: 58
Please input your guess.
  76
You guessed: 76
Too big!
```

76

`loop`

```rust
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("failed to read line");

        let guess: u32 = guess.trim().parse()
            .expect("Please type a number!");

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less    => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal   => println!("You win!"),
        }
    }
}
```

`parse()` `return`

```
$ cargo run
   Compiling guessing_game v0.1.0 (file:///home/you/projects/guessing_game)
     Running `target/guessing_game`
Guess the number!
The secret number is: 59
Please input your guess.
45
You guessed: 45
Too small!
Please input your guess.
60
You guessed: 60
Too big!
Please input your guess.
59
You guessed: 59
You win!
Please input your guess.
quit
thread '<main>' panicked at 'Please type a number!'
```

`quit`

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("failed to read line");

        let guess: u32 = guess.trim().parse()
            .expect("Please type a number!");

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less    => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal   => {
                println!("You win!");
                break;
            }
        }
    }
}
```

`You win!` `break` `main()`

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("failed to read line");

        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less    => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal   => {
                println!("You win!");
                break;
            }
        }
    }
}
```

```
let guess: u32 = match guess.trim().parse() {
    Ok(num) => num,
    Err(_) => continue,
};
```

“”“” `ok().expect()` `match` `parse()` `Result` `Ordering` `Ok` `Err` `match` `Ok(num)` `Ok` `num` `E`

```
$ cargo run
   Compiling guessing_game v0.0.1 (file:///home/you/projects/guessing_game)
     Running `target/guessing_game`
Guess the number!
The secret number is: 61
Please input your guess.
10
You guessed: 10
Too small!
Please input your guess.
99
You guessed: 99
Too big!
Please input your guess.
foo
Please input your guess.
61
You guessed: 61
You win!
```

sanguan

```rust
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("failed to read line");

        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less    => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal   => {
                println!("You win!");
                break;
            }
        }
    }
}
```

`let` `match`

Rust

Rust

> variable-bindings.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

“Hello World” Rust `let`

```
fn main() {
    let x = 5;
}
```

`fn main() {` `main()`

Patterns

Rust `let` “ ”

```
let (x, y) = (1, 2);
```

`x` 1 `y` 2.

Type annotations

Rust Rust Rust

`:`

```
let x: i32 = 5;
```

“ `x` `i32` `5` ”

`x` 32 Rust `i` `u` 81632 64

```
fn main() {
    let x = 5; // x: i32
}
```

`let` Rust Rust

Mutability

immutable

```
let x = 5;
x = 10;
```

```
error: re-assignment of immutable variable `x`
     x = 10;
     ^~~~~~~
```

`mut`

```
let mut x = 5; // mut x: i32
x = 10;
```

Rust `mut` `mut`

Rust

Initializing bindings

Rust

`src/main.rs`

```
fn main() {
    let x: i32;

    println!("Hello world!");
}
```

`cargo build` “Hello, world!”

```
   Compiling hello_world v0.0.1 (file:///home/you/projects/hello_world)
src/main.rs:2:9: 2:10 warning: unused variable: `x`, #[warn(unused_variable)] on by def
ault
src/main.rs:2     let x: i32;
                      ^
```

Rust `x`

```
fn main() {
    let x: i32;

    println!("The value of x is: {}", x);
}
```

```
$ cargo build
   Compiling hello_world v0.0.1 (file:///home/you/projects/hello_world)
src/main.rs:4:39: 4:40 error: use of possibly uninitialized variable: `x`
src/main.rs:4     println!("The value of x is: {}", x);
                                                    ^
note: in expansion of format_args!
<std macros>:2:23: 2:77 note: expansion site
<std macros>:1:1: 3:2 note: in expansion of println!
src/main.rs:4:5: 4:42 note: expansion site
error: aborting due to previous error
Could not compile `hello_world`.
```

Rust `println!`

`{}` moustachesRust *String interpolation*“” `x x`

Rust

Scope and shadowing

- `{ }` `x y` `x fn main() {}` `y`

```
fn main() {
    let x: i32 = 17;
    {
        let y: i32 = 3;
        println!("The value of x is {} and value of y is {}", x, y);
    }
    println!("The value of x is {} and value of y is {}", x, y); // This won't work
}
```

`println!` “The value of x is 17 and the value of y is 3” `println! y`

```
$ cargo build
   Compiling hello v0.1.0 (file:///home/you/projects/hello_world)
main.rs:7:62: 7:63 error: unresolved name `y`. Did you mean `x`? [E0425]
main.rs:7     println!("The value of x is {} and value of y is {}", x, y); // This won'
t work
                                                                       ^
note: in expansion of format_args!
<std macros>:2:25: 2:56 note: expansion site
<std macros>:1:1: 2:62 note: in expansion of print!
<std macros>:3:1: 3:54 note: expansion site
<std macros>:1:1: 3:58 note: in expansion of println!
main.rs:7:5: 7:65 note: expansion site
main.rs:7:62: 7:63 help: run `rustc --explain E0425` to see a detailed explanation
error: aborting due to previous error
Could not compile `hello`.

To learn more, run the command again with --verbose.
```

```
let x: i32 = 8;
{
    println!("{}", x); // Prints "8"
    let x = 12;
    println!("{}", x); // Prints "12"
}
println!("{}", x); // Prints "8"
let x =  42;
println!("{}", x); // Prints "42"
```

```
let mut x: i32 = 1;
x = 7;
let x = x; // x is now immutable and is bound to 7

let y = 4;
let y = "I can also be bound to text!"; // y is now of a different type
```

> functions.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

`main`

```
fn main() {
}
```

`fn` “” `foo`

```
fn foo() {
}
```

```
fn print_number(x: i32) {
    println!("x is: {}", x);
}
```

`print_number`

```
fn main() {
    print_number(5);
}

fn print_number(x: i32) {
    println!("x is: {}", x);
}
```

`let`

```
fn main() {
    print_sum(5, 6);
}

fn print_sum(x: i32, y: i32) {
    println!("sum is: {}", x + y);
}
```

`let`

```
fn print_sum(x, y) {
    println!("sum is: {}", x + y);
}
```

```
expected one of `!`, `:`, or `@`, found `)`
fn print_number(x, y) {
```

Haskell

```
fn add_one(x: i32) -> i32 {
    x + 1
}
```

Rust “” `- >`

```
fn add_one(x: i32) -> i32 {
    x + 1;
}
```

```
error: not all control paths return a value
fn add_one(x: i32) -> i32 {
     x + 1;
}

help: consider removing this semicolon:
     x + 1;
          ^
```

Rust “”

VS

Rust

“” `x + 1;` Rust “”“”

Ruby

```
x = y = 5
```

Rust `let`

```
let x = (let y = 5); // expected identifier, found keyword `let`
```

`let`

`y = 5` `5` Rust `()`

```
let mut y = 5;

let x = (y = 6);  // x has the value `()`, not `6`
```

RustRust Rust Rust

“”

```
fn add_one(x: i32) -> i32 {
    x + 1
}
```

`i32` `()` Rust

Early returns

Rust `return`

```
fn foo(x: i32) -> i32 {
    return x;

    // we never run this code!
    x + 1
}
```

`return`

```
fn foo(x: i32) -> i32 {
    return x + 1;
}
```

`return`

Diverging functions

Rust“”

```
fn diverges() -> ! {
    panic!("This function never returns!");
}
```

`panic!` `println!()` `println!()` `panic!()` `!` “”

`diverges()`

```
thread ‘<main>’ panicked at ‘This function never returns!’, hello.rs:2
```

`RUST_BACKTRACE` backtrace

```
$ RUST_BACKTRACE=1 ./diverges
thread '<main>' panicked at 'This function never returns!', hello.rs:2
stack backtrace:
   1:     0x7f402773a829 - sys::backtrace::write::h0942de78b6c02817K8r
   2:     0x7f402773d7fc - panicking::on_panic::h3f23f9d0b5f4c91bu9w
   3:     0x7f402773960e - rt::unwind::begin_unwind_inner::h2844b8c5e81e79558Bw
   4:     0x7f4027738893 - rt::unwind::begin_unwind::h4375279447423903650
   5:     0x7f4027738809 - diverges::h2266b4c4b850236beaa
   6:     0x7f40277389e5 - main::h19bb1149c2f00ecfBaa
   7:     0x7f402773f514 - rt::unwind::try::try_fn::h13186883479104382231
   8:     0x7f402773d1d8 - __rust_try
   9:     0x7f402773f201 - rt::lang_start::ha172a3ce74bb453aK5w
  10:     0x7f4027738a19 - main
  11:     0x7f402694ab44 - __libc_start_main
  12:     0x7f40277386c8 - <unknown>
  13:                0x0 - <unknown>
```

`RUST_BACKTRACE` Cargo `run`

```
$ RUST_BACKTRACE=1 cargo run
     Running `target/debug/diverges`
thread '<main>' panicked at 'This function never returns!', hello.rs:2
stack backtrace:
   1:     0x7f402773a829 - sys::backtrace::write::h0942de78b6c02817K8r
   2:     0x7f402773d7fc - panicking::on_panic::h3f23f9d0b5f4c91bu9w
   3:     0x7f402773960e - rt::unwind::begin_unwind_inner::h2844b8c5e81e79558Bw
   4:     0x7f4027738893 - rt::unwind::begin_unwind::h4375279447423903650
   5:     0x7f4027738809 - diverges::h2266b4c4b850236beaa
   6:     0x7f40277389e5 - main::h19bb1149c2f00ecfBaa
   7:     0x7f402773f514 - rt::unwind::try::try_fn::h13186883479104382231
   8:     0x7f402773d1d8 - __rust_try
   9:     0x7f402773f201 - rt::lang_start::ha172a3ce74bb453aK5w
  10:     0x7f4027738a19 - main
  11:     0x7f402694ab44 - __libc_start_main
  12:     0x7f40277386c8 - <unknown>
  13:                0x0 - <unknown>
```

```
# fn diverges() -> ! {
#     panic!("This function never returns!");
# }
let x: i32 = diverges();
let x: String = diverges();
```

```
let f: fn(i32) -> i32;
```

`f` `i32` `i32`

```
fn plus_one(i: i32) -> i32 {
    i + 1
}

// without type inference
let f: fn(i32) -> i32 = plus_one;

// with type inference
let f = plus_one;
```

`f`

```
# fn plus_one(i: i32) -> i32 { i + 1 }
# let f = plus_one;
let six = f(5);
```

> primitive-types.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust“”Rust

Rust `bool` `true` `false`

```
let x = true;

let y: bool = false;
```

if

`bool`

`char`

`char` Unicode `'` `char`

```
let x = 'x';
let two_hearts = '';
```

Rust `char` 1 4

`char`

Rust

`u16` 16

```
let x = 42; // x has type i32

let y = 1.0; // y has type f64
```

- i16
- i32
- i64

- i8
- u16
- u32
- u64
- u8
- isize
- usize
- f32
- f64

4 4 `-8 +7` “” 4 `0 +15`

`8 16 32 64` `u32` 32 `i64` 64

Rust “size” `isize usize`

Rust `f32 f64` IEEE-754

Rust

```
let a = [1, 2, 3]; // a: [i32; 3]
let mut m = [1, 2, 3]; // mut m: [i32; 3]
```

`[T; N]` `T` `N`

`a` `0`

```
let a = [0; 20]; // a: [i32; 20]
```

`a.len()` `a`

```
let a = [1, 2, 3];

println!("a has {} elements", a.len());
```

subscript notation

```
let names = ["Graydon", "Brian", "Niko"]; // names: [&str; 3]

println!("The second name is: {}", names[1]);
```

0 `names[0]` `names[1]` `The second name is: Brian` bug

`array`

Slices

slice“”

`&` `[]` `&` `[]`

```
let a = [0, 1, 2, 3, 4];
let middle = &a[1..4]; // A slice of a: just the elements 1, 2, and 3
let complete = &a[..]; // A slice containing all of the elements in a
```

`&[T]` `T`

`slices`

`str`

Rust `str` &str

`str`

Tuples

tuples

```
let x = (1, "hello");
```

2

```
let x: (i32, &str) = (1, "hello");
```

`i32` `&str` `&str` *string slice*

```
let mut x = (1, 2); // x: (i32, i32)
let y = (2, 3); // y: (i32, i32)

x = y;
```

letdestructuring let

```
let (x, y, z) = (1, 2, 3);

println!("x is {}", x);
```

`let` `let` `let` “”“”

```
(0,); // single-element tuple
(0); // zero in parentheses
```

Tuple Indexing

```
let tuple = (1, 2, 3);

let x = tuple.0;
let y = tuple.1;
let z = tuple.2;

println!("x is {}", x);
```

`0` `. []`

`tuple`

```
fn foo(x: i32) -> i32 { x }

let x: fn(i32) -> i32 = foo;
```

`x` “” `i32 i32`

> comments.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust *line commentsdoc comments*

```
// Line comments are anything after '//' and extend to the end of the line.

let x = 5; // this is also a line comment.

// If you have a long explanation for something, you can put line comments next
// to each other. Put a space between the // and your comment so that it's
// more readable.
```

`///` `//` Markdown

```
/// Adds one to the number given.
///
/// # Examples
///
///
```

/// let five = 5; /// /// assert_eq!(6, add_one(5)); /// ``` fn add_one(x: i32) -> i32 { x + 1 }

`//!`cratecratelib.rsmod.rs

```rust
//! # The Rust Standard Library
//!
//! The Rust Standard Library provides the essential runtime
//! functionality for building portable Rust software.
```

`assert_eq!` `panic!` `assert!` `false` `panic!`

rustdocHTML

If

> if.md
>
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust If

If

If

```
let x = 5;

if x == 5 {
    println!("x is five!");
}
```

`x` `if` `true` `false`

`false` `else`

```
let x = 5;

if x == 5 {
    println!("x is five!");
} else {
    println!("x is not five :(");
}
```

`else if`

```
let x = 5;

if x == 5 {
    println!("x is five!");
} else if x == 6 {
    println!("x is six!");
} else {
    println!("x is not five or six :(");
}
```

```rust
let x = 5;

let y = if x == 5 {
    10
} else {
    15
}; // y: i32
```

```rust
let x = 5;

let y = if x == 5 { 10 } else { 15 }; // y: i32
```

`if` `else if ()`

> loops.md
> commit 2217cf1af27d7980aba9deca4e78165cab5e80fc

Rust 3 `loop` `while` `for`

loop

`loop` Rust `loop` Rust Rust `loop`

```
loop {
    println!("Loop forever!");
}
```

while

Rust `while`

```
let mut x = 5; // mut x: u32
let mut done = false; // mut done: bool

while !done {
    x += x - 3;

    println!("{}", x);

    if x % 5 == 0 {
        done = true;
    }
}
```

`while`

```
while true {
```

`loop`

```
loop {
```

Rust `while true` `loop`

for

`for` Rust `for` Rust `for` “C” `for`

```
for (x = 0; x < 10; x++) {
    printf( "%d\n", x );
}
```

```
for x in 0..10 {
    println!("{}", x); // x: i32
}
```

```
for var in expression {
    code
}
```

. `var` `for`

`0..10` `0 9 10`

Rust “C” `for` C

Enumerate

`.enumerate()`

On ranges

```
for (i,j) in (5..10).enumerate() {
    println!("i = {} and j = {}", i, j);
}
```

```
i = 0 and j = 5
i = 1 and j = 6
i = 2 and j = 7
i = 3 and j = 8
i = 4 and j = 9
```

On iterators:

```
for (linenumber, line) in lines.enumerate() {
    println!("{}: {}", linenumber, line);
}
```

```
0: Content of line one
1: Content of line two
2: Content of line tree
3: Content of line four
```

Ending iteration early

`while`

```
let mut x = 5;
let mut done = false;

while !done {
    x += x - 3;

    println!("{}", x);

    if x % 5 == 0 {
        done = true;
    }
}
```

`mut` `done` ,Rust `break` `continue`

`break`

```
let mut x = 5;

loop {
    x += x - 3;
    println!("{}", x);
    if x % 5 == 0 { break; }
}
```

`loop` `break`

`continue`

```
for x in 0u32..10 {
    if x % 2 == 0 { continue; }

    println!("{}", x);
}
```

`break` `continue` `while` `for`

Loop labels

`break` `continue` `break` `continue` `break` `continue` `break` `continue`

`x` `y`

```
'outer: for x in 0..10 {
    'inner: for y in 0..10 {
        if x % 2 == 0 { continue 'outer; } // continues the loop over x
        if y % 2 == 0 { continue 'inner; } // continues the loop over y
        println!("x: {}, y: {}", x, y);
    }
}
```

> ownership.md
> commit fcc356373bba8c20a18d26bc81242c77c4153089

3RustRustRustRust --

-
- : "" (references)
-

33Rust

Meta

Rust *zero-cost abstractions* Rust

Rust “” Rust “” Rust Rust Rust

Ownership

Rust

```
fn foo() {
    let v = vec![1, 2, 3];
}
```

`v` Vecvector 3 `v` `foo()` Rustvectordeterministically

vector `push()`

vector `Vec<T>` `v` `Vec<i32>`

Move semantics

Rust vector

```
let v = vec![1, 2, 3];

let v2 = v;
```

`v`

```rust
let v = vec![1, 2, 3];

let v2 = v;

println!("v[0] is: {}", v[0]);
```

```
error: use of moved value: `v`
println!("v[0] is: {}", v[0]);
                        ^
```

```rust
fn take(v: Vec<i32>) {
    // what happens here isn’t important.
}

let v = vec![1, 2, 3];

take(v);

println!("v[0] is: {}", v[0]);
```

“use of moved value”“” Rust

```rust
let v = vec![1, 2, 3];

let v2 = v;
```

vector `v` `[1, 2, 3]` `v` `v2` `v2` Rust Rust `v`

`Copy`

trait `Copy` trait

```rust
let v = 1;

let v2 = v;

println!("v is: {}", v);
```

`v` `i32` `Copy` `v` `v2` , `v` `i32`

`Copy` trait“” `i32` `bool` `Copy` trait

```rust
fn main() {
    let a = 5;

    let _y = double(a);
    println!("{}", a);
}

fn double(x: i32) -> i32 {
    x * 2
}
```

```rust
fn main() {
    let a = true;

    let _y = change_truth(a);
    println!("{}", a);
}

fn change_truth(x: bool) -> bool {
    !x
}
```

`Copy` trait

```
error: use of moved value: `a`
println!("{}", a);
               ^
```

trait `Copy`

More than ownership

```rust
fn foo(v: Vec<i32>) -> Vec<i32> {
    // do stuff with v

    // hand back ownership
    v
}
```

```
fn foo(v1: Vec<i32>, v2: Vec<i32>) -> (Vec<i32>, Vec<i32>, i32) {
    // do stuff with v1 and v2

    // hand back ownership, and the result of our function
    (v1, v2, 42)
}

let v1 = vec![1, 2, 3];
let v2 = vec![1, 2, 3];

let (v1, v2, answer) = foo(v1, v2);
```

Rust trait

> references-and-borrowing.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

3 Rust Rust Rust Rust --

-
-
-

3 3 Rust

Meta

Rust *zero-cost abstractions* Rust

Rust “” Rust “” Rust Rust Rust

```rust
fn foo(v1: Vec<i32>, v2: Vec<i32>) -> (Vec<i32>, Vec<i32>, i32) {
    // do stuff with v1 and v2

    // hand back ownership, and the result of our function
    (v1, v2, 42)
}

let v1 = vec![1, 2, 3];
let v2 = vec![1, 2, 3];

let (v1, v2, answer) = foo(v1, v2);
```

Rust "

```rust
fn foo(v1: &Vec<i32>, v2: &Vec<i32>) -> i32 {
    // do stuff with v1 and v2

    // return the answer
    42
}

let v1 = vec![1, 2, 3];
let v2 = vec![1, 2, 3];

let answer = foo(&v1, &v2);

// we can use v1 and v2 here!
```

`Vec<i32>` `&Vec<i32>` `v1` `v2` `&v1` `&v2` `&T` ”“ `foo()`

`foo()`

```rust
fn foo(v: &Vec<i32>) {
     v.push(5);
}

let v = vec![];

foo(&v);
```

```
error: cannot borrow immutable borrowed content `*v` as mutable
v.push(5);
^
```

`&mut`

`&mut T` “”

```rust
let mut x = 5;
{
    let y = &mut x;
    *y += 1;
}
println!("{}", x);
```

`6` `y` `x` `y` `x` `mut`

`y` `*` `*y` `y` `&mut`

`&mut` , `{ }`

```
error: cannot borrow `x` as immutable because it is also borrowed as mutable
    println!("{}", x);
                   ^
note: previous borrow of `x` occurs here; the mutable borrow prevents
subsequent moves, borrows, or modification of `x` until the borrow ends
        let y = &mut x;
                     ^
note: previous borrow ends here
fn main() {

}
^
```

Rust

- 0 N &T
- 1 (&mut T

> 2 1 “”

2 `&mut` Rust

Thinking in scopes

```
let mut x = 5;
let y = &mut x;

*y += 1;

println!("{}", x);
```

```
error: cannot borrow `x` as immutable because it is also borrowed as mutable
    println!("{}", x);
                   ^
```

`x` `&mut T` `&T`

```
note: previous borrow ends here
fn main() {

}
^
```

`println!` Rust

```
let mut x = 5;

let y = &mut x;    // -+ &mut borrow of x starts here
                   //  |
*y += 1;           //  |
                   //  |
println!("{}", x); // -+ - try to borrow x here
                   // -+ &mut borrow of x ends here
```

`y` `&x`

```
let mut x = 5;

{
    let y = &mut x; // -+ &mut borrow starts here
    *y += 1;        //  |
}                   // -+ ... and ends here

println!("{}", x);  // <- try to borrow x here
```

Issues borrowing prevents

Iterator invalidation

“”Rust

```
let mut v = vec![1, 2, 3];

for i in &v {
    println!("{}", i);
}
```

1 3. `v`

```
let mut v = vec![1, 2, 3];

for i in &v {
    println!("{}", i);
    v.push(34);
}
```

```
error: cannot borrow `v` as mutable because it is also borrowed as immutable
    v.push(34);
    ^
note: previous borrow of `v` occurs here; the immutable borrow prevents
subsequent moves or mutable borrows of `v` until the borrow ends
for i in &v {
          ^
note: previous borrow ends here
for i in &v {
    println!(“{}”, i);
    v.push(34);
}
^
```

`v`

Rust

Rust

```
let y: &i32;
{
    let x = 5;
    y = &x;
}

println!("{}", y);
```

```
error: `x` does not live long enough
    y = &x;
         ^
note: reference must be valid for the block suffix following statement 0 at
2:16...
let y: &i32;
{
    let x = 5;
    y = &x;
}

note: ...but borrowed value is only valid for the block suffix following
statement 0 at 4:18
    let x = 5;
    y = &x;
}
```

`y` `x` `x` “”

```
let y: &i32;
let x = 5;
y = &x;

println!("{}", y);
```

```
error: `x` does not live long enough
y = &x;
     ^
note: reference must be valid for the block suffix following statement 0 at
2:16...
    let y: &i32;
    let x = 5;
    y = &x;

    println!("{}", y);
}

note: ...but borrowed value is only valid for the block suffix following
statement 1 at 3:14
    let x = 5;
    y = &x;

    println!("{}", y);
}
```

`y` `x` `y` `x`

> lifetimes.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

3 Rust Rust Rust Rust --

-
- : "" (references)
-

3 3 Rust

Meta

Rust *zero-cost abstractions* Rust

Rust “” Rust “” Rust Rust Rust

-
-
-
-

dangling pointer“”

Rust *lifetime*

```
// implicit
fn foo(x: &i32) {
}

// explicit
fn bar<'a>(x: &'a i32) {
}
```

`'a` “ a”

```
fn bar<'a>(...)
```

`<>` `<>` “”

`<>` `bar 'a`

```
fn bar<'a, 'b>(...)
```

```
...(x: &'a i32)
```

`&mut`

```
...(x: &'a mut i32)
```

`&mut i32` `&'a mut i32` `& mut i32 'a` `&mut i32` “ `i32` ” `&'a mut i32` “'ai32
”

`struct`

```
struct Foo<'a> {
    x: &'a i32,
}

fn main() {
    let y = &5; // this is the same as `let _y = 5; let y = &_y;`
    let f = Foo { x: y };

    println!("{}", f.x);
}
```

`struct`

```
struct Foo<'a> {
# x: &'a i32,
# }
```

```
# struct Foo<'a> {
x: &'a i32,
# }
```

`Foo i32`

`impl`

`Foo`

```rust
struct Foo<'a> {
    x: &'a i32,
}

impl<'a> Foo<'a> {
    fn x(&self) -> &'a i32 { self.x }
}

fn main() {
    let y = &5; // this is the same as `let _y = 5; let y = &_y;`
    let f = Foo { x: y };

    println!("x is: {}", f.x());
}
```

`impl` `Foo` `'a` `impl<'a>` `'a` `Foo<'a>`

```rust
fn x_or_y<'a>(x: &'a str, y: &'a str) -> &'a str {
#     x
# }
```

`x` `y` `x` `y`

```rust
fn x_or_y<'a, 'b>(x: &'a str, y: &'b str) -> &'a str {
#     x
# }
```

`x` `y` `x`

Thinking in scopes

```rust
fn main() {
    let y = &5;     // -+ y goes into scope
                    //  |
    // stuff        //  |
                    //  |
}                   // -+ y goes out of scope
```

`Foo`

```rust
struct Foo<'a> {
    x: &'a i32,
}

fn main() {
    let y = &5;           // -+ y goes into scope
    let f = Foo { x: y }; // -+ f goes into scope
    // stuff              //  |
                          //  |
}                         // -+ f and y go out of scope
```

`f` `y`

```rust
struct Foo<'a> {
    x: &'a i32,
}

fn main() {
    let x;                    // -+ x goes into scope
                              //  |
    {                         //  |
        let y = &5;           // ---+ y goes into scope
        let f = Foo { x: y }; // ---+ f goes into scope
        x = &f.x;             //  | | error here
    }                         // ---+ f and y go out of scope
                              //  |
    println!("{}", x);        //  |
}                             // -+ x goes out of scope
```

`f` `y` `x` `x = &f.x` `x`

'static

`static` Rust `'static`

```rust
let x: &'static str = "Hello, world.";
```

`&'static str`

```rust
static FOO: i32 = 5;
let x: &'static i32 = &FOO;
```

`i32` `x`

Lifetime Elision

Rust“” 3

> *input lifetime and output lifetime.*

```
fn foo<'a>(bar: &'a str)
```

```
fn foo<'a>() -> &'a str
```

```
fn foo<'a>(bar: &'a str) -> &'a str
```

3

-
-
- `&self` `&mut self` `self`

```
fn print(s: &str); // elided
fn print<'a>(s: &'a str); // expanded

fn debug(lvl: u32, s: &str); // elided
fn debug<'a>(lvl: u32, s: &'a str); // expanded

// In the preceding example, `lvl` doesn’t need a lifetime because it’s not a
// reference (`&`). Only things relating to references (such as a `struct`
// which contains a reference) need lifetimes.

fn substr(s: &str, until: u32) -> &str; // elided
fn substr<'a>(s: &'a str, until: u32) -> &'a str; // expanded

fn get_str() -> &str; // ILLEGAL, no inputs

fn frob(s: &str, t: &str) -> &str; // ILLEGAL, two inputs
fn frob<'a, 'b>(s: &'a str, t: &'b str) -> &str; // Expanded: Output lifetime is unclear


fn get_mut(&mut self) -> &mut T; // elided
fn get_mut<'a>(&'a mut self) -> &'a mut T; // expanded

fn args<T:ToCStr>(&mut self, args: &[T]) -> &mut Command // elided
fn args<'a, 'b, T:ToCStr>(&'a mut self, args: &'b [T]) -> &'a mut Command // expanded

fn new(buf: &mut [u8]) -> BufWriter; // elided
fn new<'a>(buf: &'a mut [u8]) -> BufWriter<'a> // expanded
```

> mutability.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust

```
let x = 5;
x = 6; // error!
```

`mut`

```
let mut x = 5;

x = 6; // no problem!
```

`x` `i32`

```
let mut x = 5;
let y = &mut x;
```

`y` `y` `y = &mut z` `y` `*y = 5`

```
let mut x = 5;
let mut y = &mut x;
```

`y`

`mut`

```
let (mut x, y) = (5, 6);

fn foo(mut x: i32) {
```

VS Interior vs. Exterior Mutability

Rust“”“” Arc

```
use std::sync::Arc;

let x = Arc::new(5);
let y = x.clone();
```

`clone()` `Arc<T>` `mut x` `&mut 5`

RustRust

> - 0N `&T`
> - 1 `&mut T`

“” `Arc<T>` `clone()` `&T` `&mut T`

std::cell

```
use std::cell::RefCell;

let x = RefCell::new(42);

let y = x.borrow_mut();
```

`RefCell` `borrow_mut()` `&mut`

```
use std::cell::RefCell;

let x = RefCell::new(42);

let y = x.borrow_mut();
let z = x.borrow_mut();
```

`RefCell` Rust `panic!` Rust

Field-level mutability

`&mut` `&mut`

```
struct Point {
    x: i32,
    mut y: i32, // nope
}
```

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

let mut a = Point { x: 5, y: 6 };

a.x = 10;

let b = Point { x: 5, y: 6};

b.x = 10; // error: cannot assign to immutable field `b.x`
```

`Cell<T>`

```rust
use std::cell::Cell;

struct Point {
    x: i32,
    y: Cell<i32>,
}

let mut point = Point { x: 5, y: Cell::new(6) };

point.y.set(7);

println!("y: {:?}", point.y);
```

`y: Cell { value: 7 }` `y`

> structs.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

2D `x` `y`

```
let origin_x = 0;
let origin_y = 0;
```

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

fn main() {
    let origin = Point { x: 0, y: 0 }; // origin: Point

    println!("The origin is at ({}, {})", origin.x, origin.y);
}
```

`struct` `PointInSpace` `Point_In_Space`

`let` `key: value`

`origin.x`

Rust `mut`

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

fn main() {
    let mut point = Point { x: 0, y: 0 };

    point.x = 5;

    println!("The point is at ({}, {})", point.x, point.y);
}
```

`The point is at (5, 0)`

Rust

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

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

fn main() {
    let mut point = Point { x: 0, y: 0 };

    point.x = 5;

    let point = point; // this new binding can’t change now

    point.y = 6; // this causes an error
}
```

`&mut`

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

struct PointRef<'a> {
    x: &'a mut i32,
    y: &'a mut i32,
}

fn main() {
    let mut point = Point { x: 0, y: 0 };

    {
        let r = PointRef { x: &mut point.x, y: &mut point.y };

        *r.x = 5;
        *r.y = 6;
    }

    assert_eq!(5, point.x);
    assert_eq!(6, point.y);
}
```

Update syntax

`..` `struct`

```
struct Point3d {
    x: i32,
    y: i32,
    z: i32,
}

let mut point = Point3d { x: 0, y: 0, z: 0 };
point = Point3d { y: 1, .. point };
```

`point` `y` `x` `z` `struct`

```
# struct Point3d {
#     x: i32,
#     y: i32,
#     z: i32,
# }
let origin = Point3d { x: 0, y: 0, z: 0 };
let point = Point3d { z: 1, x: 2, .. origin };
```

Rust `struct`

```
struct Color(i32, i32, i32);
struct Point(i32, i32, i32);

let black = Color(0, 0, 0);
let origin = Point(0, 0, 0);
```

`black` `origin`

```
let black = Color(0, 0, 0);
let origin = Point(0, 0, 0);
```

`Color` `Point`

```rust
struct Color {
    red: i32,
    blue: i32,
    green: i32,
}

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

newtype

```rust
struct Inches(i32);

let length = Inches(10);

let Inches(integer_length) = length;
println!("length is {} inches", integer_length);
```

`let` `let Inches(integer_length)` `integer_length` `10`

Unit-like structs

```rust
struct Electron;

let x = Electron;
```

“” `()` “”

> enums.md
> commit 31e39cd05c9b28c78b087aa9314f246b0b0b5cfa

Rust `enum`

```
enum Message {
    Quit,
    ChangeColor(i32, i32, i32),
    Move { x: i32, y: i32 },
    Write(String),
}
```

`enum` “”

`::` `enum`

```
let x: Message = Message::Move { x: 3, y: 4 };

enum BoardGameTurn {
    Move { squares: i32 },
    Pass,
}

let y: BoardGameTurn = BoardGameTurn::Move { squares: 1 };
```

`Move`

“”“”

```
fn process_color_change(msg: Message) {
    let Message::ChangeColor(r, g, b) = msg; // compile-time error
}
```

`match` Rust

Constructors as functions

```
# enum Message {
# Write(String),
# }
let m = Message::Write("Hello, world".to_string());
```

```
# enum Message {
# Write(String),
# }
fn foo(x: String) -> Message {
    Message::Write(x)
}

let x = foo("Hello, world".to_string());
```

`String` vector `Message::Write` vector

```
# enum Message {
# Write(String),
# }

let v = vec!["Hello".to_string(), "World".to_string()];

let v1: Vec<Message> = v.into_iter().map(Message::Write).collect();
```

> match.md
> commit fc4bb5f77060b5822f25edbabbdf7a1d48a7f8fe

`if` / `else` `else` Rust `match` `if` / `else`

```
let x = 5;

match x {
    1 => println!("one"),
    2 => println!("two"),
    3 => println!("three"),
    4 => println!("four"),
    5 => println!("five"),
    _ => println!("something else"),
}
```

`match` `val => expression` `match` “” `match`

`match` *exhaustiveness checking*Rust

```
error: non-exhaustive patterns: `_` not covered
```

Rust `x` 32 1 2,147,483,647 `_` `match` 1 5 `mtach` `x` 6 `_`

`match` `let`

```
let x = 5;

let number = match x {
    1 => "one",
    2 => "two",
    3 => "three",
    4 => "four",
    5 => "five",
    _ => "something else",
};
```

Matching on enums

`match`

```
enum Message {
    Quit,
    ChangeColor(i32, i32, i32),
    Move { x: i32, y: i32 },
    Write(String),
}

fn quit() { /* ... */ }
fn change_color(r: i32, g: i32, b: i32) { /* ... */ }
fn move_cursor(x: i32, y: i32) { /* ... */ }

fn process_message(msg: Message) {
    match msg {
        Message::Quit => quit(),
        Message::ChangeColor(r, g, b) => change_color(r, g, b),
        Message::Move { x: x, y: y } => move_cursor(x, y),
        Message::Write(s) => println!("{}", s),
    };
}
```

Rust `_`

`match` `if` if let `match`

> patterns.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust

`_` “”

```
let x = 1;

match x {
    1 => println!("one"),
    2 => println!("two"),
    3 => println!("three"),
    _ => println!("anything"),
}
```

`one`

```
let x = 1;
let c = 'c';

match c {
    x => println!("x: {} c: {}", x, c),
}

println!("x: {}", x)
```

```
x: c c: c
x: 1
```

`x =>` `x` `c` `x` `x` `x` `x`

Multiple patterns

`|`

```
let x = 1;

match x {
    1 | 2 => println!("one or two"),
    3 => println!("three"),
    _ => println!("anything"),
}
```

```
one or two
```

Destructuring

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

let origin = Point { x: 0, y: 0 };

match origin {
    Point { x: x, y: y } => println!("({},{})", x, y),
}
```

`:`

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

let origin = Point { x: 0, y: 0 };

match origin {
    Point { x: x1, y: y1 } => println!("({},{})", x1, y1),
}
```

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

let origin = Point { x: 0, y: 0 };

match origin {
    Point { x: x, .. } => println!("x is {}", x),
}
```

`x is 0`

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

let origin = Point { x: 0, y: 0 };

match origin {
    Point { y: y, .. } => println!("y is {}", y),
}
```

`y is 0`

“”

Ignoring bindings

`_` `Result<T, E>` `match`

```
# let some_value: Result<i32, &'static str> = Err("There was an error");
match some_value {
    Ok(value) => println!("got a value: {}", value),
    Err(_) => println!("an error occurred"),
}
```

`Ok` `value` `Err` `_`

`_`

```
fn coordinate() -> (i32, i32, i32) {
    // generate and return some sort of triple tuple
}

let (x, _, z) = coordinate();
```

`x` `z`

`..`

```
enum OptionalTuple {
    Value(i32, i32, i32),
    Missing,
}

let x = OptionalTuple::Value(5, -2, 3);

match x {
    OptionalTuple::Value(..) => println!("Got a tuple!"),
    OptionalTuple::Missing => println!("No such luck."),
}
```

```
Got a tuple!
```

`ref` `ref mut`

`ref`

```
let x = 5;

match x {
    ref r => println!("Got a reference to {}", r),
}
```

```
Got a reference to 5
```

`match` `r` `&i32` `ref` `ref mut`

```
let mut x = 5;

match x {
    ref mut mr => println!("Got a mutable reference to {}", mr),
}
```

Ranges

`...`

```
let x = 1;

match x {
    1 ... 5 => println!("one through five"),
    _ => println!("anything"),
}
```

`one through five`

`char`

```
let x = '';

match x {
    'a' ... 'j' => println!("early letter"),
    'k' ... 'z' => println!("late letter"),
    _ => println!("something else"),
}
```

`something else`

`@`

```
let x = 1;

match x {
    e @ 1 ... 5 => println!("got a range element {}", e),
    _ => println!("anything"),
}
```

`got a range element 1`

```
#[derive(Debug)]
struct Person {
    name: Option<String>,
}

let name = "Steve".to_string();
let mut x: Option<Person> = Some(Person { name: Some(name) });
match x {
    Some(Person { name: ref a @ Some(_), .. }) => println!("{:?}", a),
    _ => {}
}
```

`Some("Steve")` Person `name` `a`

`|` `@`

```
let x = 5;

match x {
    e @ 1 ... 5 | e @ 8 ... 10 => println!("got a range element {}", e),
    _ => println!("anything"),
}
```

Guards

`if` *match guards*

```
enum OptionalInt {
    Value(i32),
    Missing,
}

let x = OptionalInt::Value(5);

match x {
    OptionalInt::Value(i) if i > 5 => println!("Got an int bigger than five!"),
    OptionalInt::Value(..) => println!("Got an int!"),
    OptionalInt::Missing => println!("No such luck."),
}
```

`Got an int!`

`if` `if`

```
let x = 4;
let y = false;

match x {
    4 | 5 if y => println!("yes"),
    _ => println!("no"),
}
```

`no` `if` `4 | 5` `5` `if`

```
(4 | 5) if y => ...
```

```
4 | (5 if y) => ...
```

Mix and Match

()

```
match x {
    Foo { x: Some(ref name), y: None } => ...
}
```

> method-syntax.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

```
baz(bar(foo(x)));
```

“baz bar foo”“foo bar baz”

```
x.foo().bar().baz();
```

Rust `impl` *method call syntax*

```
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}

fn main() {
    let c = Circle { x: 0.0, y: 0.0, radius: 2.0 };
    println!("{}", c.area());
}
```

`12.566371`

`impl` `area`

`&self` 3 `self` `&self` `&mut self` `x.foo()` `x` 3 `x` 3 `self` `&self` `&mut self` `area` `&self` `Circle` `radius`

`&self`

```rust
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn reference(&self) {
       println!("taking self by reference!");
    }

    fn mutable_reference(&mut self) {
       println!("taking self by mutable reference!");
    }

    fn takes_ownership(self) {
       println!("taking ownership of self!");
    }
}
```

`impl`

```rust
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn reference(&self) {
       println!("taking self by reference!");
    }
}

impl Circle {
    fn mutable_reference(&mut self) {
       println!("taking self by mutable reference!");
    }
}

impl Circle {
    fn takes_ownership(self) {
       println!("taking ownership of self!");
    }
}
```

Chaining method calls

`foo.bar()` `foo.bar().baz()` “” `self`

```
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }

    fn grow(&self) -> Circle {
        Circle { x: self.x, y: self.y, radius: (self.radius * 10.0) }
    }
}

fn main() {
    let c = Circle { x: 0.0, y: 0.0, radius: 2.0 };
    println!("{}", c.area());

    let d = c.grow().area();
    println!("{}", d);
}
```

```
fn grow(&self) -> Circle {
```

`Circle`

Associated functions

`self` Rust

```rust
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn new(x: f64, y: f64, radius: f64) -> Circle {
        Circle {
            x: x,
            y: y,
            radius: radius,
        }
    }
}

fn main() {
    let c = Circle::new(0.0, 0.0, 2.0);
}
```

associated function `Circle` `Struct::method()` `ref.method()`

Builder Pattern

`x` `y` `0.0` `radius` `1.0` Rust

```rust
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}

struct CircleBuilder {
    coordinate: f64,
    radius: f64,
}

impl CircleBuilder {
    fn new() -> CircleBuilder {
        CircleBuilder { coordinate: 0.0, radius: 0.0, }
    }

    fn coordinate(&mut self, coordinate: f64) -> &mut CircleBuilder {
    self.coordinate = coordinate;
    self
    }

    fn radius(&mut self, radius: f64) -> &mut CircleBuilder {
    self.radius = radius;
    self
    }

    fn finalize(&self) -> Circle {
        Circle { x: self.coordinate, y: self.coordinate, radius: self.radius }
    }
}

fn main() {
    let c = CircleBuilder::new()
                .coordinate(10.0)
                .radius(5.0)
                .finalize();


    println!("area: {}", c.area());
}
```

`CircleBuilder` `Circle` `area()` `CircleBuilder: finalize()` `Circle` `CircleBuilder` `Circle`

Vectors

> vectors.md
> commit 5b9dd6a016adb5ed67e150643fb7e21dcc916845

“Vector”“” Vec `<T>` vectorvector `String` `&str` `vec!`

```
let v = vec![1, 2, 3]; // v: Vec<i32>
```

`println!` `vec!` `[]` Rust

`vec!`

```
let v = vec![0; 10]; // ten zeroes
```

vector `[]`

```
let v = vec![1, 2, 3, 4, 5];

println!("The third element of v is {}", v[2]);
```

`0` 3 `v[2]`

`usize`

```
let v = vec![1, 2, 3, 4, 5];

let i: usize = 0;
let j: i32 = 0;

// works
v[i];

// doesn’t
v[j];
```

`usize`

```
error: the trait `core::ops::Index<i32>` is not implemented for the type
`collections::vec::Vec<_>` [E0277]
v[j];
^~~~
note: the type `collections::vec::Vec<_>` cannot be indexed by `i32`
error: aborting due to previous error
```

`i32`

Out-of-bounds Access

```
let v = vec![1, 2, 3];
println!("Item 7 is {}", v[7]);
```

panic

```
thread '<main>' panicked at 'index out of bounds: the len is 3 but the index is 7'
```

panic `get` `get_mut` `None`

```
let v = vec![1, 2, 3];
match v.get(7) {
    Some(x) => println!("Item 7 is {}", x),
    None => println!("Sorry, this vector is too short.")
}
```

vector `for` 3

```
let mut v = vec![1, 2, 3, 4, 5];

for i in &v {
    println!("A reference to {}", i);
}

for i in &mut v {
    println!("A mutable reference to {}", i);
}

for i in v {
    println!("Take ownership of the vector and its element {}", i);
}
```

vector vectorAPI

> strings.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

RustRustC

UTF-8UnicodeUTF-8nullnull

Rust `&str` `String` `&str` *string slices* `&'static str`

```
let greeting = "Hello there."; // greeting: &'static str
```

`"Hello there."` `&'static str` `greeting`

```
let s = "foo
    bar";

assert_eq!("foo\n        bar", s);
```

`\`

```
let s = "foo\
    bar";

assert_eq!("foobar", s);
```

Rust `&str` `String` UTF-8 `String` `to_string`

```
let mut s = "Hello".to_string(); // mut s: String
println!("{}", s);

s.push_str(", world.");
println!("{}", s);
```

`String` `&` `&str`

```
fn takes_slice(slice: &str) {
    println!("Got: {}", slice);
}

fn main() {
    let s = "Hello".to_string();
    takes_slice(&s);
}
```

`&str` trait `&str` TcpStream::connect `ToSocketAddrs` `&str  String  &*`

```
use std::net::TcpStream;

TcpStream::connect("192.168.0.1:3000"); // &str parameter

let addr_string = "192.168.0.1:3000".to_string();
TcpStream::connect(&*addr_string); // convert addr_string to &str
```

`String  &str` `&str  String`

Indexing

UTF-8

```
let s = "hello";

println!("The first letter of s is {}", s[0]); // ERROR!!!
```

`[]` UTF-8NUnicode“”codepoints

```
let hachiko = ""     ;

for b in hachiko.as_bytes() {
print!("{}, ", b);
}

println!("");

for c in hachiko.chars() {
print!("{}, ", c);
}

println!("");
```

```
229, 191, 160, 231, 138, 172, 227, 131, 143, 227, 131, 129, 229, 133, 172,
, , , , ,
```

`char`

```
let dog = hachiko.chars().nth(1); // kinda like hachiko[1]
```

`char`

Slicing

```
let dog = "hachiko";
let hachi = &dog[0..5];
```

```
let dog = ""      ;
let hachi = &dog[0..2];
```

```
thread '<main>' panicked at 'index 0 and/or 2 in ``      do not lie on
character boundary'
```

Concatenation

`String  &str`

```
let hello = "Hello ".to_string();
let world = "world!";

let hello_world = hello + world;
```

`String  &`

```
let hello = "Hello ".to_string();
let world = "world!".to_string();

let hello_world = hello + &world;
```

`&String  &str  Deref`

> generics.md

Rust *parametric polymorphism*parametric `poly` `morph`

Rust `Option<T>`

```
enum Option<T> {
    Some(T),
    None,
}
```

`<T>` `T` `Option<T>`

```
let x: Option<i32> = Some(5);
```

`Option<i32>` `Option<T>` `Option` `T` `i32` `Some(T)` `T` `5` `i32`

```
let x: Option<f64> = Some(5);
// error: mismatched types: expected `core::option::Option<f64>`,
// found `core::option::Option<_>` (expected f64 but found integral variable)
```

`f64` `Option<T>`

```
let x: Option<i32> = Some(5);
let y: Option<f64> = Some(5.0f64);
```

Rust `Result<T, E>`

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

`T` `E` `Result<T, E>`

```
enum Result<A, Z> {
    Ok(A),
    Err(Z),
}
```

`T` `type` `E` `error` Rust

`Result<T, E>`

```
fn takes_anything<T>(x: T) {
    // do something with x
}
```

`<T>` “” `x: T` “ `x` `T` ”

```
fn takes_two_of_the_same_things<T>(x: T, y: T) {
    // ...
}
```

```
fn takes_two_things<T, U>(x: T, y: U) {
    // ...
}
```

Generic structs

`struct`

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

let int_origin = Point { x: 0, y: 0 };
let float_origin = Point { x: 0.0, y: 0.0 };
```

`<T>` `x: T`

`struct` `impl`

```
# struct Point<T> {
#     x: T,
#     y: T,
# }
#
impl<T> Point<T> {
    fn swap(&mut self) {
        std::mem::swap(&mut self.x, &mut self.y);
    }
}
```

`Option<T>` `Vec<T>` trait bound

Traits

> traits.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

trait Rust

`impl`

```
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}
```

trait trait trait `Circle  HasArea` trait

```
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

trait HasArea {
    fn area(&self) -> f64;
}

impl HasArea for Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}
```

`trait  impl` `impl` trait `impl Trait for Item  impl Item`

trait boundTrait bounds on generic functions

trait bound

```
fn print_area<T>(shape: T) {
    println!("This shape has an area of {}", shape.area());
}
```

Rust

```
error: no method named `area` found for type `T` in the current scope
```

`T` `area` `T` trait bound

```
fn print_area<T: HasArea>(shape: T) {
    println!("This shape has an area of {}", shape.area());
}
```

`<T: HasArea>` `any type that implements the HasArea trait` `HasArea` trait trait `HasArea` `.area()`

```
trait HasArea {
    fn area(&self) -> f64;
}

struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl HasArea for Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}

struct Square {
    x: f64,
    y: f64,
    side: f64,
}

impl HasArea for Square {
    fn area(&self) -> f64 {
        self.side * self.side
    }
}

fn print_area<T: HasArea>(shape: T) {
    println!("This shape has an area of {}", shape.area());
}

fn main() {
    let c = Circle {
        x: 0.0f64,
        y: 0.0f64,
        radius: 1.0f64,
    };

    let s = Square {
        x: 0.0f64,
        y: 0.0f64,
        side: 1.0f64,
    };

    print_area(c);
    print_area(s);
}
```

```
This shape has an area of 3.141593
This shape has an area of 1
```

`print_area`

```
print_area(5);
```

```
error: the trait `HasArea` is not implemented for the type `_` [E0277]
```

trait boundTrait bounds on generic structs

trait bound bound `Rectangle<T>` `is_square()`

```
struct Rectangle<T> {
    x: T,
    y: T,
    width: T,
    height: T,
}

impl<T: PartialEq> Rectangle<T> {
    fn is_square(&self) -> bool {
        self.width == self.height
    }
}

fn main() {
    let mut r = Rectangle {
        x: 0,
        y: 0,
        width: 47,
        height: 47,
    };

    assert!(r.is_square());

    r.height = 42;
    assert!(!r.is_square());
}
```

`is_square()` `core::cmp::PartialEq` trait

```
impl<T: PartialEq> Rectangle<T> { ... }
```

`Rectangle` ———— `HasArea` `Square` `Circle` trait

trait Rules for implementing traits

trait trait `i32` `HasArea`

```
trait HasArea {
    fn area(&self) -> f64;
}

impl HasArea for i32 {
    fn area(&self) -> f64 {
        println!("this is silly");

        *self as f64
    }
}

5.area();
```

Wild West trait I/O `Write` trait `File` `File`

```
let mut f = std::fs::File::open("foo.txt").ok().expect("Couldn’t open foo.txt");
let buf = b"whatever"; // byte string literal. buf: &[u8; 8]
let result = f.write(buf);
```

```
error: type `std::fs::File` does not implement any method in scope named `write`
let result = f.write(buf);
               ^~~~~~~~~~
```

`use` `Write` trait

```
use std::io::Write;

let mut f = std::fs::File::open("foo.txt").expect("Couldn’t open foo.txt");
let buf = b"whatever";
let result = f.write(buf);
# result.unwrap(); // ignore the error
```

`int` `use` trait

traittrait `impl` `i32` `HasArea` trait `HasArea` `i32` `Float` traitRusttrait

traittrait *monomorphization*monomorph *statically dispatched* trait

trait boundMultiple trait bounds

trait

```
fn foo<T: Clone>(x: T) {
    x.clone();
}
```

1 `+`

```
use std::fmt::Debug;

fn foo<T: Clone + Debug>(x: T) {
    x.clone();
    println!("{:?}", x);
}
```

`T Clone Debug`

where Where clause

trait

```
use std::fmt::Debug;

fn foo<T: Clone, K: Clone + Debug>(x: T, y: K) {
    x.clone();
    y.clone();
    println!("{:?}", y);
}
```

Rust“where ”

```
use std::fmt::Debug;

fn foo<T: Clone, K: Clone + Debug>(x: T, y: K) {
    x.clone();
    y.clone();
    println!("{:?}", y);
}

fn bar<T, K>(x: T, y: K) where T: Clone, K: Clone + Debug {
    x.clone();
    y.clone();
    println!("{:?}", y);
}

fn main() {
    foo("Hello", "world");
    bar("Hello", "workd");
}
```

`foo()` `bar()` `where` `where`

```
use std::fmt::Debug;

fn bar<T, K>(x: T, y: K)
    where T: Clone,
          K: Clone + Debug {

    x.clone();
    y.clone();
    println!("{:?}", y);
}
```

`where`

```
trait ConvertTo<Output> {
    fn convert(&self) -> Output;
}

impl ConvertTo<i64> for i32 {
    fn convert(&self) -> i64 { *self as i64 }
}

// can be called with T == i32
fn normal<T: ConvertTo<i64>>(x: &T) -> i64 {
    x.convert()
}

// can be called with T == i64
fn inverse<T>() -> T
        // this is using ConvertTo as if it were "ConvertFrom<i32>"
        where i32: ConvertTo<T> {
    42.convert()
}
```

`where`   `i32`   `T`

Default methods

trait

```
trait Foo {
    fn is_valid(&self) -> bool;

    fn is_invalid(&self) -> bool { !self.is_valid() }
}
```

`Foo` trait `is_valid()` `is_invalid()`

```
# trait Foo {
#     fn is_valid(&self) -> bool;
#
#     fn is_invalid(&self) -> bool { !self.is_valid() }
# }
struct UseDefault;

impl Foo for UseDefault {
    fn is_valid(&self) -> bool {
        println!("Called UseDefault.is_valid.");
        true
    }
}

struct OverrideDefault;

impl Foo for OverrideDefault {
    fn is_valid(&self) -> bool {
        println!("Called OverrideDefault.is_valid.");
        true
    }

    fn is_invalid(&self) -> bool {
        println!("Called OverrideDefault.is_invalid!");
        true // overrides the expected value of is_invalid()
    }
}

let default = UseDefault;
assert!(!default.is_invalid()); // prints "Called UseDefault.is_valid."

let over = OverrideDefault;
assert!(over.is_invalid()); // prints "Called OverrideDefault.is_invalid!"
```

Inheritance

traittrait

```
trait Foo {
    fn foo(&self);
}

trait FooBar : Foo {
    fn foobar(&self);
}
```

`FooBar` `Foo`

```
struct Baz;

impl Foo for Baz {
    fn foo(&self) { println!("foo"); }
}

impl FooBar for Baz {
    fn foobar(&self) { println!("foobar"); }
}
```

`Foo` Rust

```
error: the trait `main::Foo` is not implemented for the type `main::Baz` [E0277]
```

Deriving

`Debug` `Default` trait Rust Rust trait

```
#[derive(Debug)]
struct Foo;

fn main() {
    println!("{:?}", Foo);
}
```

deriving trait

- Clone
- Copy
- Debug
- Default
- Eq
- Hash
- Ord
- PartialEq
- PartialOrd

`Drop`

> drop.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

trait Rust trait `Drop` `Drop` trait

```rust
struct HasDrop;

impl Drop for HasDrop {
    fn drop(&mut self) {
        println!("Dropping!");
    }
}

fn main() {
    let x = HasDrop;

    // do stuff

} // x goes out of scope here
```

`main()` `x` `Drop` `Drop` `drop()` `self`

`Drop` dropped

```rust
struct Firework {
    strength: i32,
}

impl Drop for Firework {
    fn drop(&mut self) {
        println!("BOOM times {}!!!", self.strength);
    }
}

fn main() {
    let firecracker = Firework { strength: 1 };
    let tnt = Firework { strength: 100 };
}
```

```
BOOM times 100!!!
BOOM times 1!!!
```

`tnt` `firecracker` TNT

`Drop` `Drop` `struct` `Arc<T>` `Drop` 0

if let

> if-let.md
> commit 797a0bd1c13175398aa0e2e45f6dbb61bcb8c329

`if let` `if` `let`

`Option<T>` `Some<T>` `None`

```rust
# let option = Some(5);
# fn foo(x: i32) { }
match option {
    Some(x) => { foo(x) },
    None => {},
}
```

`match` `if`

```rust
# let option = Some(5);
# fn foo(x: i32) { }
if option.is_some() {
    let x = option.unwrap();
    foo(x);
}
```

`if let`

```rust
# let option = Some(5);
# fn foo(x: i32) { }
if let Some(x) = option {
    foo(x);
}
```

`else`

```rust
# let option = Some(5);
# fn foo(x: i32) { }
# fn bar() { }
if let Some(x) = option {
    foo(x);
} else {
    bar();
}
```

`while let`

`while let` `while let`

```
let mut v = vec![1, 3, 5, 7, 11];
loop {
    match v.pop() {
        Some(x) =>  println!("{}", x),
        None => break,
    }
}
```

```
let mut v = vec![1, 3, 5, 7, 11];
while let Some(x) = v.pop() {
    println!("{}", x);
}
```

trait

> trait-objects.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

“”dispatch Rust “trait ”

trait `Foo` `String`

```
trait Foo {
    fn method(&self) -> String;
}
```

`u8` `String` trait

```
impl Foo for u8 {
    fn method(&self) -> String { format!("u8: {}", *self) }
}

impl Foo for String {
    fn method(&self) -> String { format!("string: {}", *self) }
}
```

trait

```
fn do_something<T: Foo>(x: T) {
    x.method();
}

fn main() {
    let x = 5u8;
    let y = "Hello".to_string();

    do_something(x);
    do_something(y);
}
```

Rust “” Rust `u8` `String` `do_something()` `do_something` Rust

```
# trait Foo { fn method(&self) -> String; }
# impl Foo for u8 { fn method(&self) -> String { format!("u8: {}", *self) } }
# impl Foo for String { fn method(&self) -> String { format!("string: {}", *self) } }
fn do_something_u8(x: u8) {
    x.method();
}

fn do_something_string(x: String) {
    x.method();
}

fn main() {
    let x = 5u8;
    let y = "Hello".to_string();

    do_something_u8(x);
    do_something_string(y);
}
```

“”code bloat

“” `#[inline]` `#[inline(always)]`

Rust “trait ” `&Foo` `Box<Foo>` trait trait

`trait` (casting) `&x as &Foo` (coercing it) `&x` `&Foo` trait

trait `&mut Foo` `&mut T` `Box<Foo>` `Box<T>`

“”trait“”type erasure

trait trait casting

```
# trait Foo { fn method(&self) -> String; }
# impl Foo for u8 { fn method(&self) -> String { format!("u8: {}", *self) } }
# impl Foo for String { fn method(&self) -> String { format!("string: {}", *self) } }

fn do_something(x: &Foo) {
    x.method();
}

fn main() {
    let x = 5u8;
    do_something(&x as &Foo);
}
```

by concercing

```rust
# trait Foo { fn method(&self) -> String; }
# impl Foo for u8 { fn method(&self) -> String { format!("u8: {}", *self) } }
# impl Foo for String { fn method(&self) -> String { format!("string: {}", *self) } }

fn do_something(x: &Foo) {
    x.method();
}

fn main() {
    let x = "Hello".to_string();
    do_something(&x);
}
```

trait `Foo`

Rust size Rust

`Foo` `String` 24 `u8` 1crate `Foo`

trait sizesize

Representation

trait “”

trait

trait `std::raw` trait

```rust
# mod foo {
pub struct TraitObject {
    pub data: *mut (),
    pub vtable: *mut (),
}
# }
```

trait“”“” `&Foo`

trait `T` `T` `Foo`

`trait_object.method()`

```
struct FooVtable {
    destructor: fn(*mut ()),
    size: usize,
    align: usize,
    method: fn(*const ()) -> String,
}

// u8:

fn call_method_on_u8(x: *const ()) -> String {
    // the compiler guarantees that this function is only called
    // with `x` pointing to a u8
    let byte: &u8 = unsafe { &*(x as *const u8) };

    byte.method()
}

static Foo_for_u8_vtable: FooVtable = FooVtable {
    destructor: /* compiler magic */,
    size: 1,
    align: 1,

    // cast to a function pointer
    method: call_method_on_u8 as fn(*const ()) -> String,
};


// String:

fn call_method_on_String(x: *const ()) -> String {
    // the compiler guarantees that this function is only called
    // with `x` pointing to a String
    let string: &String = unsafe { &*(x as *const String) };

    string.method()
}

static Foo_for_String_vtable: FooVtable = FooVtable {
    destructor: /* compiler magic */,
    // values for a 64-bit computer, halve them for 32-bit ones
    size: 24,
    align: 8,

    method: call_method_on_String as fn(*const ()) -> String,
};
```

`destructor` `u8` `String` `Box<Foo>` trait `Box` `size` `align` trait

`Foo` `Foo` trait

```
let a: String = "foo".to_string();
let x: u8 = 1;

// let b: &Foo = &a;
let b = TraitObject {
    // store the data
    data: &a,
    // store the methods
    vtable: &Foo_for_String_vtable
};

// let y: &Foo = x;
let y = TraitObject {
    // store the data
    data: &x,
    // store the methods
    vtable: &Foo_for_u8_vtable
};

// b.method();
(b.vtable.method)(b.data);

// y.method();
(y.vtable.method)(y.data);
```

Object Safety

trait trait vector `Clone` trait

```
let v = vec![1, 2, 3];
let o = &v as &Clone;
```

```
error: cannot convert to a trait object because trait `core::clone::Clone` is not objec
t-safe [E0038]
let o = &v as &Clone;
        ^~
note: the trait cannot require that `Self : Sized`
let o = &v as &Clone;
        ^~
```

`Clone` “object-safe” trait trait trait

- trait `Self: Sized`
-

  `Self: Sized`
-

- `Self`

  `Self` “ trait `Self` ”

> closures.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

free variables “” Rust

```
let plus_one = |x: i32| x + 1;

assert_eq!(2, plus_one(1));
```

`plus_one` `|` `x + 1` `{}`

```
let plus_two = |x| {
    let mut result: i32 = x;

    result += 1;
    result += 1;

    result
};

assert_eq!(4, plus_two(2));
```

`fn`

```
let plus_one = |x: i32| -> i32 { x + 1 };

assert_eq!(2, plus_one(1));
```

“”“”

```
fn  plus_one_v1   (x: i32 ) -> i32 { x + 1 }
let plus_one_v2 = |x: i32 | -> i32 { x + 1 };
let plus_one_v3 = |x: i32 |          x + 1  ;
```

“”“”close over their environment

```
let num = 5;
let plus_num = |x: i32| x + num;

assert_eq!(10, plus_num(5));
```

`plus_num` `let` `num`

```
let mut num = 5;
let plus_num = |x: i32| x + num;

let y = &mut num;
```

```
error: cannot borrow `num` as mutable because it is also borrowed as immutable
    let y = &mut num;
                 ^~~
note: previous borrow of `num` occurs here due to use in closure; the immutable
  borrow prevents subsequent moves or mutable borrows of `num` until the borrow
  ends
    let plus_num = |x| x + num;
                   ^~~~~~~~~~~
note: previous borrow ends here
fn main() {
    let mut num = 5;
    let plus_num = |x| x + num;

    let y = &mut num;
}
^
```

`num`

```
let mut num = 5;
{
    let plus_num = |x: i32| x + num;

} // plus_num goes out of scope, borrow of num ends

let y = &mut num;
```

Rust

```
let nums = vec![1, 2, 3];

let takes_nums = || nums;

println!("{:?}", nums);
```

```
note: `nums` moved into closure environment here because it has type
  `[closure(()) -> collections::vec::Vec<i32>]`, which is non-copyable
let takes_nums = || nums;
                    ^~~~~~~
```

`Vec<T>` `nums` `nums`

`move`

`move`

```
let num = 5;

let owns_num = move |x: i32| x + num;
```

`move` `5 Copy owns_num 5`

```
let mut num = 5;

{
    let mut add_num = |x: i32| num += x;

    add_num(5);
}

assert_eq!(10, num);
```

`num` `add_num` `add_num mut`

`move`

```
let mut num = 5;

{
    let mut add_num = move |x: i32| num += x;

    add_num(5);
}

assert_eq!(5, num);
```

`5` `num`

`move` `move` `move` `move`

Rust

Rust trait traittrait

`()` `foo()` Rusttraittrait

```
# mod foo {
pub trait Fn<Args> : FnMut<Args> {
    extern "rust-call" fn call(&self, args: Args) -> Self::Output;
}

pub trait FnMut<Args> : FnOnce<Args> {
    extern "rust-call" fn call_mut(&mut self, args: Args) -> Self::Output;
}

pub trait FnOnce<Args> {
    type Output;

    extern "rust-call" fn call_once(self, args: Args) -> Self::Output;
}
# }
```

trait `self` `Fn` `&self` `FnMut` `&mut self` `FnOnce` `self` 3 `self` 3 trait 1

`||` `{}` 3 trait Rust `impl` trait

Taking closures as arguments

trait trait

```
fn call_with_one<F>(some_closure: F) -> i32
    where F : Fn(i32) -> i32 {

    some_closure(1)
}

let answer = call_with_one(|x| x + 2);

assert_eq!(3, answer);
```

`|x| x + 2` `call_with_one` `1`

`call_with_one`

```
fn call_with_one<F>(some_closure: F) -> i32
#     where F : Fn(i32) -> i32 {
#     some_closure(1) }
```

`F` `i32`

```
# fn call_with_one<F>(some_closure: F) -> i32
    where F : Fn(i32) -> i32 {
#   some_closure(1) }
```

`Fn` trait `i32` `i32` `Fn(i32) -> i32`

traitRust

trait

```
fn call_with_one(some_closure: &Fn(i32) -> i32) -> i32 {
    some_closure(1)
}

let answer = call_with_one(&|x| x + 2);

assert_eq!(3, answer);
```

trait `&Fn` `call_with_one` `&||`

```
fn call_with_one(some_closure: &Fn(i32) -> i32) -> i32 {
    some_closure(1)
}

fn add_one(i: i32) -> i32 {
    i + 1
}

let f = add_one;

let answer = call_with_one(&f);

assert_eq!(2, answer);
```

`f`

```
let answer = call_with_one(&add_one);
```

Returning closures

```
fn factory() -> (Fn(i32) -> i32) {
    let num = 5;

    |x| x + num
}

let f = factory();

let answer = f(1);
assert_eq!(6, answer);
```

```
error: the trait `core::marker::Sized` is not implemented for the type
`core::ops::Fn(i32) -> i32` [E0277]
fn factory() -> (Fn(i32) -> i32) {
                ^~~~~~~~~~~~~~~~
note: `core::ops::Fn(i32) -> i32` does not have a constant size known at compile-time
fn factory() -> (Fn(i32) -> i32) {
                ^~~~~~~~~~~~~~~~
error: the trait `core::marker::Sized` is not implemented for the type `core::ops::Fn(i
32) -> i32` [E0277]
let f = factory();
    ^
note: `core::ops::Fn(i32) -> i32` does not have a constant size known at compile-time
let f = factory();
    ^
```

Rust `Fn` trait(size) `Fn` (size)

```
fn factory() -> &(Fn(i32) -> i32) {
    let num = 5;

    |x| x + num
}

let f = factory();

let answer = f(1);
assert_eq!(6, answer);
```

```
error: missing lifetime specifier [E0106]
fn factory() -> &(Fn(i32) -> i32) {
                ^~~~~~~~~~~~~~~~~
```

`factory()` `'static`

```
fn factory() -> &'static (Fn(i32) -> i32) {
    let num = 5;

    |x| x + num
}

let f = factory();

let answer = f(1);
assert_eq!(6, answer);
```

```
error: mismatched types:
 expected `&'static core::ops::Fn(i32) -> i32`,
    found `[closure <anon>:7:9: 7:20]`
(expected &-ptr,
    found closure) [E0308]
         |x| x + num
         ^~~~~~~~~~~
```

`&'static Fn(i32) -> i32` `[closure <anon>:7:9: 7:20]`

`struct` `Fn` Rust `closure <anon>`

`'static` `Box` `Fn` trait

```
fn factory() -> Box<Fn(i32) -> i32> {
    let num = 5;

    Box::new(|x| x + num)
}
# fn main() {
let f = factory();

let answer = f(1);
assert_eq!(6, answer);
# }
```

```
error: closure may outlive the current function, but it borrows `num`,
which is owned by the current function [E0373]
Box::new(|x| x + num)
         ^~~~~~~~~~~
```

`5` `num`

```
fn factory() -> Box<Fn(i32) -> i32> {
    let num = 5;

    Box::new(move |x| x + num)
}
# fn main() {
let f = factory();

let answer = f(1);
assert_eq!(6, answer);
# }
```

`move Fn` `Box`

> ufcs.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

```
trait Foo {
    fn f(&self);
}

trait Bar {
    fn f(&self);
}

struct Baz;

impl Foo for Baz {
    fn f(&self) { println!("Baz’s impl of Foo"); }
}

impl Bar for Baz {
    fn f(&self) { println!("Baz’s impl of Bar"); }
}

let b = Baz;
```

`b.f()`

```
error: multiple applicable methods in scope [E0034]
b.f();
  ^~~
note: candidate #1 is defined in an impl of the trait `main::Foo` for the type
`main::Baz`
    fn f(&self) { println!("Baz’s impl of Foo"); }
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: candidate #2 is defined in an impl of the trait `main::Bar` for the type
`main::Baz`
    fn f(&self) { println!("Baz’s impl of Bar"); }
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

“”universal function call syntax

```
# trait Foo {
#     fn f(&self);
# }
# trait Bar {
#     fn f(&self);
# }
# struct Baz;
# impl Foo for Baz {
#     fn f(&self) { println!("Baz’s impl of Foo"); }
# }
# impl Bar for Baz {
#     fn f(&self) { println!("Baz’s impl of Bar"); }
# }
# let b = Baz;
Foo::f(&b);
Bar::f(&b);
```

```
Foo::
Bar::
```

traits `Foo` `Bar` Rusttrait

```
f(&b)
```

`b.f()` `f()` `&self` Rust `b` `&self` Rust `&b`

Angle-bracket Form

```
Trait::method(args);
```

```
<Type as Trait>::method(args);
```

`<>::` `<>` `Type as Trait` `method` `Trait` `as Trait`

```
trait Foo {
    fn clone(&self);
}

#[derive(Clone)]
struct Bar;

impl Foo for Bar {
    fn clone(&self) {
        println!("Making a clone of Bar");

        <Bar as Clone>::clone(self);
    }
}
```

trait

> crates-and-modules.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust

Rust *cratemodule* *(library)(package)*“Cargo”RustCargo

root module

phrases“”“”

```
                             +-----------+
                         +---| greetings |
                         |   +-----------+
           +---------+   |
           | english |---+
           +---------+   |   +-----------+
           |             +---| farewells |
+---------+  |               +-----------+
| phrases |---+
+---------+  |                +-----------+
           |              +---| greetings |
           +----------+   |   +-----------+
           | japanese |---+
           +----------+   |
                          |   +-----------+
                          +---| farewells |
                              +-----------+
```

`phrases` `phrases`

Cargo

```
$ cargo new phrases
$ cd phrases
```

```
$ tree .
.
├── Cargo.toml
└── src
    └── lib.rs

1 directory, 2 files
```

`src/lib.rs` `phrases`

`mod` `src/lib.rs`

```rust
// in src/lib.rs

mod english {
    mod greetings {

    }

    mod farewells {

    }
}

mod japanese {
    mod greetings {

    }

    mod farewells {

    }
}
```

`mod` Rust `lower_snake_case` `{}`

`mod` `mod` `::` 4 `english::greetings` `english::farewells` `japanese::greetings` `japanese::farewells` `english::greetings` `japanese::greetings` `greetings`

`lib.rs` `main()` Cargo

```
$ cargo build
   Compiling phrases v0.0.1 (file:///home/you/projects/phrases)
$ ls target
deps  libphrases-a7448e02a0468eaa.rlib  native
```

`libphrase-hash.rlib`

Rust

```rust
mod english {
    // contents of our module go here
}
```

```rust
mod english;
```

Rust `english.rs` `english/mod.rs`

`mod`

```
$ tree .
.
├── Cargo.lock
├── Cargo.toml
├── src
│   ├── english
│   │   ├── farewells.rs
│   │   ├── greetings.rs
│   │   └── mod.rs
│   ├── japanese
│   │   ├── farewells.rs
│   │   ├── greetings.rs
│   │   └── mod.rs
│   └── lib.rs
└── target
    └── debug
        ├── build
        ├── deps
        ├── examples
        ├── libphrases-a7448e02a0468eaa.rlib
        └── native
```

`src/lib.rs`

```rust
mod english;
mod japanese;
```

Rust `src/english.rs` `src/japanese.rs` `src/english/mod.rs` `src/japanese/mod.rs`

`src/english/mod.rs` `src/japanese/mod.rs`

```rust
mod greetings;
mod farewells;
```

Rust `src/english/greetings.rs` `src/japanese/greetings.rs` `src/english/farewells/mod.rs` `src/japanese/farewells/mod.rs` `src/english/greetings.rs` `src/japanese/farewells.rs`

`src/english/greetings.rs` `src/japanese/farewells.rs`

`src/english/greetings.rs`

```
fn hello() -> String {
    "Hello!".to_string()
}
```

`src/english/farewells.rs`

```
fn goodbye() -> String {
    "Goodbye.".to_string()
}
```

`src/japanese/greetings.rs`

```
fn hello() -> String {
    ""     .to_string()
}
```

“konnichiwa”

`src/japanese/farewells.rs`

```
fn goodbye() -> String {
    ""     .to_string()
}
```

“Sayōnara”

`src/main.rs`

```
extern crate phrases;

fn main() {
    println!("Hello in English: {}", phrases::english::greetings::hello());
    println!("Goodbye in English: {}", phrases::english::farewells::goodbye());

    println!("Hello in Japanese: {}", phrases::japanese::greetings::hello());
    println!("Goodbye in Japanese: {}", phrases::japanese::farewells::goodbye());
}
```

`extern crate` Rust `phrases` `phrases`

“like-this” crate Rust `extern crate like_this;`

Cargo `src/main.rs` `src/lib.rs` `src/main.rs`

4

```
$ cargo build
   Compiling phrases v0.0.1 (file:///home/you/projects/phrases)
src/main.rs:4:38: 4:72 error: function `hello` is private
src/main.rs:4     println!("Hello in English: {}", phrases::english::greetings::hello()
);
                                                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in expansion of format_args!
<std macros>:2:25: 2:58 note: expansion site
<std macros>:1:1: 2:62 note: in expansion of print!
<std macros>:3:1: 3:54 note: expansion site
<std macros>:1:1: 3:58 note: in expansion of println!
phrases/src/main.rs:4:5: 4:76 note: expansion site
```

Rust

Rust `pub` `english` `src/main.rs`

```
extern crate phrases;

fn main() {
    println!("Hello in English: {}", phrases::english::greetings::hello());
    println!("Goodbye in English: {}", phrases::english::farewells::goodbye());
}
```

`src/lib.rs` `english` `pub`

```
pub mod english;
mod japanese;
```

`src/english/mod.rs` `pub`

```
pub mod greetings;
pub mod farewells;
```

`src/english/greetings.rs` `fn` `pub`

```
pub fn hello() -> String {
    "Hello!".to_string()
}
```

`src/english/farewells.rs`

```
pub fn goodbye() -> String {
    "Goodbye.".to_string()
}
```

`japanese`

```
$ cargo run
   Compiling phrases v0.0.1 (file:///home/you/projects/phrases)
src/japanese/greetings.rs:1:1: 3:2 warning: function is never used: `hello`, #[warn(dea
d_code)] on by default
src/japanese/greetings.rs:1 fn hello() -> String {
src/japanese/greetings.rs:2     ""     .to_string()
src/japanese/greetings.rs:3 }
src/japanese/farewells.rs:1:1: 3:2 warning: function is never used: `goodbye`, #[warn(d
ead_code)] on by default
src/japanese/farewells.rs:1 fn goodbye() -> String {
src/japanese/farewells.rs:2     ""     .to_string()
src/japanese/farewells.rs:3 }
     Running `target/debug/phrases`
Hello in English: Hello!
Goodbye in English: Goodbye.
```

`phrases::english::greetings::hello()` Rust `use`

`use`

Rust `use` `src/main.rs`

```
extern crate phrases;

use phrases::english::greetings;
use phrases::english::farewells;

fn main() {
    println!("Hello in English: {}", greetings::hello());
    println!("Goodbye in English: {}", farewells::goodbye());
}
```

`use`

```
extern crate phrases;

use phrases::english::greetings::hello;
use phrases::english::farewells::goodbye;

fn main() {
    println!("Hello in English: {}", hello());
    println!("Goodbye in English: {}", goodbye());
}
```

Rust `japanese`

```
extern crate phrases;

use phrases::english::greetings::hello;
use phrases::japanese::greetings::hello;

fn main() {
    println!("Hello in English: {}", hello());
    println!("Hello in Japanese: {}", hello());
}
```

Rust

```
   Compiling phrases v0.0.1 (file:///home/you/projects/phrases)
src/main.rs:4:5: 4:40 error: a value named `hello` has already been imported in this mo
dule [E0252]
src/main.rs:4 use phrases::japanese::greetings::hello;
                  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
error: aborting due to previous error
Could not compile `phrases`.
```

Rust

```
use phrases::english::greetings;
use phrases::english::farewells;
```

```
use phrases::english::{greetings, farewells};
```

`pub use`

`use`

`src/main.rs`

```
extern crate phrases;

use phrases::english::{greetings,farewells};
use phrases::japanese;

fn main() {
    println!("Hello in English: {}", greetings::hello());
    println!("Goodbye in English: {}", farewells::goodbye());

    println!("Hello in Japanese: {}", japanese::hello());
    println!("Goodbye in Japanese: {}", japanese::goodbye());
}
```

`src/lib.rs` `japanese`

```
pub mod english;
pub mod japanese;
```

`src/japanese/greetings.rs`

```
pub fn hello() -> String {
    ""     .to_string()
}
```

`src/japanese/farewells.rs`

```
pub fn goodbye() -> String {
    ""     .to_string()
}
```

`src/japanese/mod.rs`

```
pub use self::greetings::hello;
pub use self::farewells::goodbye;

mod greetings;
mod farewells;
```

`pub use` `japanese` `pub use` `phrases::japanese::hello()` `phrases::japanese::goodbye()` `phrases::japanese::greetings::hello()` `phrases::japanese::farewells::goodby`

`japanese` `pub use` `greetings` `pub use self::greetings::*`

`self` `use` `self` `use` `use super::` `self` `.` `super` `..` shell

`use` `foo::bar()` `foo` `::` `foo`

`pub use` `mod` Rust `use`

```
$ cargo run
   Compiling phrases v0.0.1 (file:///home/you/projects/phrases)
     Running `target/debug/phrases`
Hello in English: Hello!
Goodbye in English: Goodbye.
Hello in Japanese:
Goodbye in Japanese:
```

Rust `extern crate` `use`

```
extern crate phrases as sayings;

use sayings::japanese::greetings as ja_greetings;
use sayings::japanese::farewells::*;
use sayings::english::{self, greetings as en_greetings, farewells as en_farewells};

fn main() {
    println!("Hello in English; {}", en_greetings::hello());
    println!("And in Japanese: {}", ja_greetings::hello());
    println!("Goodbye in English: {}", english::farewells::goodbye());
    println!("Again: {}", en_farewells::goodbye());
    println!("And in Japanese: {}", goodbye());
}
```

`extern crate` `use` crate “phrases”“sayings” `use` crate `japanese::greetings` `ja_greetings` `greetings`

`use` `sayings::japanese::farewells` `goodbye`

`use` “brace expansion” `use` Linux shell

```
use sayings::english;
use sayings::english::greetings as en_greetings;
use sayings::english::farewells as en_farewells;
```

`use` `self`

`const` `static`

> const-and-static.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust `const`

```
const N: i32 = 5;
```

let `const`

Rust

`static`

Rust“”

```
static N: i32 = 5;
```

let `static`

`'static`

```
static NAME: &'static str = "Steve";
```

`mut`

```
static mut N: i32 = 5;
```

`N` `static mut` unsafe `unsafe`

```
unsafe {
    N += 1;

    println!("N: {}", N);
}
```

`static Sync`

`const` `static`

Which construct should I use?

`const`

C `#define` “C#definestatic”Rust

> attributes.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust“”

```
#[test]
# fn foo() {}
```

```
# mod foo {
#![test]
# }
```

`!`

```
#[foo]
struct Foo;

mod bar {
    #![bar]
}
```

`#[foo]` `struct` `#![bar]` `mod`

```
#[test]
fn check() {
    assert_eq!(2, 1 + 1);
}
```

`#[test]`

```
#[inline(always)]
fn super_fast_fn() {
# }
```

```
#[cfg(target_os = "macos")]
mod macos_only {
# }
```

Rust Rust

`type`

> type-aliases.md
> commit 63bb3e66ee559d7e02f877a05a6bc54c9a5ab0d5

`type`

```
type Name = String;
```

```
type Name = String;

let x: Name = "Hello".to_string();
```

Rust

```
let x: i32 = 5;
let y: i64 = 5;

if x == y {
   // ...
}
```

```
error: mismatched types:
 expected `i32`,
    found `i64`
(expected i32,
    found i64) [E0308]
     if x == y {
             ^
```

```
type Num = i32;

let x: i32 = 5;
let y: Num = 5;

if x == y {
   // ...
}
```

`Num` `i32`

```rust
use std::result;

enum ConcreteError {
    Foo,
    Bar,
}

type Result<T> = result::Result<T, ConcreteError>;
```

`Result` `ConcreteError` `Result<T, E>` `E` `io::Result`

> casting-between-types.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust `as` `transmute` Rust

Coercion

`as`

`let const static`

- `&mut T &T`

- `*mut T *const T`

- `&T *const T`
- `&mut T *mut T`

`Deref`

`as`

`as`

```
let x: i32 = 5;

let y = x as i64;
```

`e as U1 as U2` `e as U2` `U1 U2`

Explicit coercions

`e as U e T T U`

`e as U`

- `e` `T` `T` `U` `numeric-cast`
- `e` C `U` `enum-cast`
- `e` `bool` `char` `T` `prim-int-cast`
- `e` `u8` `U` `char` `u8-char-cast`

```
let one = true as u8;
let at_sign = 64 as char;
let two_hundred = -56i8 as u8;
```

- `i32` -> `u32` `no-op`
- `u32` -> `u8`
- `u8` -> `u32`
  - zero-extend
  - sign-extend
- 0
  - Undefined Behavior Inf NaN bug
-
- f32 f64
- f64 f32
  - f32 Undefined Behavior bug

```
let a = 300 as *const char; // a pointer to location 300
let b = a as u32;
```

`e as U`

- `e` `*T` `U` `*U_0` `U_0: Sized` `unsize_kind(T) == unsize_kind(U_0)` `ptr-ptr-cast`
- `e` `*T` `U` `T: Sized` `ptr-addr-cast`
- `e` `U` `*U_0` `U_0: Sized` `addr-ptr-cast`
- `e` `&[T; n]` `U` `*const T` `array-ptr-cast`
- `e` `U` `*T` `T: Sized` `fptr-ptr-cast`
- `e` `U` `fptr-addr-cast`

`transmute`

`as` 4 `u32`

```
let a = [0u8, 0u8, 0u8, 0u8];

let b = a as u32; // four eights makes 32
```

```
error: non-scalar cast: `[u8; 4]` as `u32`
let b = a as u32; // four eights makes 32
        ^~~~~~~~
```

“non-scalar cast”

`transmute` Rust

4 `u8` `u32` `transmute` `as` Rust

```
use std::mem;

unsafe {
    let a = [0u8, 0u8, 0u8, 0u8];

    let b = mem::transmute::<[u8; 4], u32>(a);
}
```

`unsafe` `mem::transmute` `a` `unsafe`

`transmute`

```
use std::mem;

unsafe {
    let a = [0u8, 0u8, 0u8, 0u8];

    let b = mem::transmute::<[u8; 4], u64>(a);
}
```

```
error: transmute called on types with different sizes: [u8; 4] (32 bits) to u64
(64 bits)
```

> associated-types.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust‘’ `Graph` traittrait `Graph<N, E>`

```
trait Graph<N, E> {
    fn has_edge(&self, &N, &N) -> bool;
    fn edges(&self, &N) -> Vec<E>;
    // etc
}
```

`Graph` `N` ode `E` dge

```
fn distance<N, E, G: Graph<N, E>>(graph: &G, start: &N, end: &N) -> u32 { ... }
```

`Edge` `E`

`Graph` `N` ode `E` dge

```
trait Graph {
    type N;
    type E;

    fn has_edge(&self, &Self::N, &Self::N) -> bool;
    fn edges(&self, &Self::N) -> Vec<Self::E>;
    // etc
}
```

`Graph`

```
fn distance<G: Graph>(graph: &G, start: &G::N, end: &G::N) -> uint { ... }
```

`E` dge

`Graph` trait

```
trait Graph {
    type N;
    type E;

    fn has_edge(&self, &Self::N, &Self::N) -> bool;
    fn edges(&self, &Self::N) -> Vec<Self::E>;
}
```

`type` trait

`type` `N` `Display`

```
use std::fmt;

trait Graph {
    type N: fmt::Display;
    type E;

    fn has_edge(&self, &Self::N, &Self::N) -> bool;
    fn edges(&self, &Self::N) -> Vec<Self::E>;
}
```

trait trait `impl` `Graph`

```
# trait Graph {
#     type N;
#     type E;
#     fn has_edge(&self, &Self::N, &Self::N) -> bool;
#     fn edges(&self, &Self::N) -> Vec<Self::E>;
# }
struct Node;

struct Edge;

struct MyGraph;

impl Graph for MyGraph {
    type N = Node;
    type E = Edge;

    fn has_edge(&self, n1: &Node, n2: &Node) -> bool {
        true
    }

    fn edges(&self, n: &Node) -> Vec<Edge> {
        Vec::new()
    }
}
```

`true` `Vec<Edge>` trait 3 `struct` `struct` 3

`impl` trait

`=` trait `=` `impl`

trait

traittrait

```
# trait Graph {
#     type N;
#     type E;
#     fn has_edge(&self, &Self::N, &Self::N) -> bool;
#     fn edges(&self, &Self::N) -> Vec<Self::E>;
# }
# struct Node;
# struct Edge;
# struct MyGraph;
# impl Graph for MyGraph {
#     type N = Node;
#     type E = Edge;
#     fn has_edge(&self, n1: &Node, n2: &Node) -> bool {
#         true
#     }
#     fn edges(&self, n: &Node) -> Vec<Edge> {
#         Vec::new()
#     }
# }
let graph = MyGraph;
let obj = Box::new(graph) as Box<Graph>;
```

```
error: the value of the associated type `E` (from the trait `main::Graph`) must
be specified [E0191]
let obj = Box::new(graph) as Box<Graph>;
          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
24:44 error: the value of the associated type `N` (from the trait
`main::Graph`) must be specified [E0191]
let obj = Box::new(graph) as Box<Graph>;
          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

trait

```rust
# trait Graph {
#     type N;
#     type E;
#     fn has_edge(&self, &Self::N, &Self::N) -> bool;
#     fn edges(&self, &Self::N) -> Vec<Self::E>;
# }
# struct Node;
# struct Edge;
# struct MyGraph;
# impl Graph for MyGraph {
#     type N = Node;
#     type E = Edge;
#     fn has_edge(&self, n1: &Node, n2: &Node) -> bool {
#         true
#     }
#     fn edges(&self, n: &Node) -> Vec<Edge> {
#         Vec::new()
#     }
# }
let graph = MyGraph;
let obj = Box::new(graph) as Box<Graph<N=Node, E=Edge>>;
```

`N=Node` `Node` `N` `E=Edge` `impl` trait

> unsized-types.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

`i32` 324“”“” `[T]` `t`

Rust

1. `&[T]` `[T]` `&[T]` `[T]`
2.
3. `struct`

`[T]`

```
impl Foo for str {
```

```
impl<T> Foo for [T] {
```

```
impl Foo for &str {
```

`impl for str` bug `impl`

`?Sized`

`?Sized`

```
struct Foo<T: ?Sized> {
    f: T,
}
```

`?` “ `T` `Sized` ” `T` `T: Sized` `?`

> operators-and-overloading.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust

`+` `Add`

```
use std::ops::Add;

#[derive(Debug)]
struct Point {
    x: i32,
    y: i32,
}

impl Add for Point {
    type Output = Point;

    fn add(self, other: Point) -> Point {
        Point { x: self.x + other.x, y: self.y + other.y }
    }
}

fn main() {
    let p1 = Point { x: 1, y: 0 };
    let p2 = Point { x: 2, y: 3 };

    let p3 = p1 + p2;

    println!("{:?}", p3);
}
```

`main` `Point` `+` `Point` `Add<Output=Point>`

`std::ops`

`Add`

```
# mod foo {
pub trait Add<RHS = Self> {
    type Output;

    fn add(self, rhs: RHS) -> Self::Output;
}
# }
```

3 `impl Add` `RHS` `Self` `Output` `let z = x + y` `x` `Self` `y` `RHS` `z` `Self::Output`

```
# struct Point;
# use std::ops::Add;
impl Add<i32> for Point {
    type Output = f64;

    fn add(self, rhs: i32) -> f64 {
        // add an i32 to a Point and get an f64
# 1.0
    }
}
```

```
let p: Point = // ...
let x: f64 = p + 2i32;
```

trait

trait trait `HasArea` trait `Square`

```
use std::ops::Mul;

trait HasArea<T> {
    fn area(&self) -> T;
}

struct Square<T> {
    x: T,
    y: T,
    side: T,
}

impl<T> HasArea<T> for Square<T>
        where T: Mul<Output=T> + Copy {
    fn area(&self) -> T {
        self.side * self.side
    }
}

fn main() {
    let s = Square {
        x: 0.0f64,
        y: 0.0f64,
        side: 12.0f64,
    };

    println!("Area of s: {}", s.area());
}
```

`HasArea` `Square` `T` `f64` `impl`

```rust
impl<T> HasArea<T> for Square<T>
        where T: Mul<Output=T> + Copy { ... }
```

`area` `T` `std::ops::Mul` `Add` `Mul` `Output` `T` `T` Rust `self.side`

Deref

> deref-coercions.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

`Deref` `*`

```
use std::ops::Deref;

struct DerefExample<T> {
    value: T,
}

impl<T> Deref for DerefExample<T> {
    type Target = T;

    fn deref(&self) -> &T {
        &self.value
    }
}

fn main() {
    let x = DerefExample { value: 'a' };
    assert_eq!('a', *x);
}
```

`Deref` “deref coercions” `U` `Deref<Target=T>` `&U` `&T`

```
fn foo(s: &str) {
    // borrow a string for a second
}

// String implements Deref<Target=str>
let owned = "Hello".to_string();

// therefore, this works:
foo(&owned);
```

`&` `owned` `String` `&owned` `&String` `impl Deref<Target=str> for String` `&String` `&str` `foo()`

Rust `Rc<T>` `Deref<Target=T>`

```rust
use std::rc::Rc;

fn foo(s: &str) {
    // borrow a string for a second
}

// String implements Deref<Target=str>
let owned = "Hello".to_string();
let counted = Rc::new(owned);

// therefore, this works:
foo(&counted);
```

`String` `Rc<T>` `Rc<String>` `String` `foo` `Rc<String>` `String` `String` `&str`
Rust

```rust
fn foo(s: &[i32]) {
    // borrow a slice for a second
}

// Vec<T> implements Deref<Target=[T]>
let owned = vec![1, 2, 3];

foo(&owned);
```

`Deref`

`Deref`

`Deref` `T` `&T` Rust

```rust
struct Foo;

impl Foo {
    fn foo(&self) { println!("Foo"); }
}

let f = Foo;

f.foo();
```

`f` `foo` `&self`

```rust
f.foo();
(&f).foo();
(&&f).foo();
(&&&&&&&&f).foo();
```

`&&&&&&&&&&&&&&&&Foo Foo` `*` `* Deref`

> macros.md
> commit ccaa7e5146ba0ee47d3b7301121a05da6e484f49

Rust

RustRust

“”Rust

Rust

“”Rust

`vec!` vector

```
let x: Vec<u32> = vec![1, 2, 3];
# assert_eq!(x, [1, 2, 3]);
```

```
let x: Vec<u32> = {
    let mut temp_vec = Vec::new();
    temp_vec.push(1);
    temp_vec.push(2);
    temp_vec.push(3);
    temp_vec
};
# assert_eq!(x, [1, 2, 3]);
```

actual

actual `vec!` libcollections

```
macro_rules! vec {
    ( $( $x:expr ),* ) => {
        {
            let mut temp_vec = Vec::new();
            $(
                temp_vec.push($x);
            )*
            temp_vec
        }
    };
}
# fn main() {
#     assert_eq!(vec![1,2,3], [1, 2, 3]);
# }
```

```
macro_rules! vec { ... }
```

`vec` `fn vec` `vec` `vec!`

```
( $( $x:expr ),* ) => { ... };
```

`match` Rust `=>` “” *matcher*

`$x:expr` Rust `$x` `expr` *fragment specifier* `$(...)` `*` 0

Rust

```
macro_rules! foo {
    (x => $e:expr) => (println!("mode X: {}", $e));
    (y => $e:expr) => (println!("mode Y: {}", $e));
}

fn main() {
    foo!(y => 3);
}
```

```
mode Y: 3
```

```
foo!(z => 3);
```

```
error: no rules expected the token `z`
```

Rust

```
$(
    temp_vec.push($x);
)*
```

`$x` `push` “”

`$x` `:expr`

`vec!`

```
macro_rules! foo {
    () => {{
        ...
    }}
}
```

`macro_rules!` `()` `[]`

`vec!` `let`

Repetition

1. `$(...)*` `$name` “”
2. `$name` `$(...)*`

```
macro_rules! o_O {
    (
        $(
            $x:expr; [ $( $y:expr ),* ]
        );*
    ) => {
        &[ $($( $x + $y ),*),* ]
    }
}

fn main() {
    let a: &[i32]
        = o_O!(10; [1, 2, 3];
               20; [4, 5, 6]);

    assert_eq!(a, [11, 12, 13, 24, 25, 26]);
}
```

`$(...)*` “0” `$(...)+` “1” `+ *`

Macro-by-ExamplePDF

Hygiene

C `13 25`

```
#define FIVE_TIMES(x) 5 * x

int main() {
    printf("%d\n", FIVE_TIMES(2 + 3));
    return 0;
}
```

`5 * 2 + 3` CRust

```
macro_rules! five_times {
    ($x:expr) => (5 * $x);
}

fn main() {
    assert_eq!(25, five_times!(2 + 3));
}
```

`$x`

*variable capture*C GNU C Rust

```
#define LOG(msg) ({ \
    int state = get_log_state(); \
    if (state > 0) { \
        printf("log(%d): %s\n", state, msg); \
    } \
})
```

```
const char *state = "reticulating splines";
LOG(state)
```

```
const char *state = "reticulating splines";
int state = get_log_state();
if (state > 0) {
    printf("log(%d): %s\n", state, state);
}
```

`state`

Rust

```
macro_rules! log {
    ($msg:expr) => {{
        let state: i32 = get_log_state();
        if state > 0 {
            println!("log({}): {}", state, $msg);
        }
    }};
}

fn main() {
    let state: &str = "reticulating splines";
    log!(state);
}
```

Rust*syntax context* `main` `state` `state` “”

```
macro_rules! foo {
    () => (let x = 3);
}

fn main() {
    foo!();
    println!("{}", x);
}
```

```
macro_rules! foo {
    ($v:ident) => (let $v = 3);
}

fn main() {
    foo!(x);
    println!("{}", x);
}
```

`let` loop items

```
macro_rules! foo {
    () => (fn x() { });
}

fn main() {
    foo!();
    x();
}
```

HTML

```
# #![allow(unused_must_use)]
macro_rules! write_html {
    ($w:expr, ) => (());

    ($w:expr, $e:tt) => (write!($w, "{}", $e));

    ($w:expr, $tag:ident [ $($inner:tt)* ] $($rest:tt)*) => {{
        write!($w, "<{}>", stringify!($tag));
        write_html!($w, $($inner)*);
        write!($w, "</{}>", stringify!($tag));
        write_html!($w, $($rest)*);
    }};
}

fn main() {
#   // FIXME(#21826)
    use std::fmt::Write;
    let mut out = String::new();

    write_html!(&mut out,
        html[
            head[title["Macros guide"]]
            body[h1["Macros are the best!"]]
        ]);

    assert_eq!(out,
        "<html><head><title>Macros guide</title></head>\
         <body><h1>Macros are the best!</h1></body></html>");
}
```

`rustc --pretty expanded` `rustc` `--pretty expanded` `--pretty expanded,hygiene`

`rustc` feature gates

- `log_syntax!(...)` “”
- `trace_macros!(true)` `trace_macros!(false)`

Rust Rust

Rust

- 0
- 0
-
-

-

/Rust

- `foo! { ... }`
- `foo!(...);`

Rust `foo!([)` Rust

token trees

- `()` `[]` `{}`
-

fragment specifier

- `ident` `x` `foo`
- `path` `T::SpecialA`
- `expr` `2 + 2` `if true then { 1 } else { 2 }` `f(42)`
- `ty` `i32` `Vec<(char, String)>` `&T`
- `pat` `Some(t)` `(17, 'a')` `_`
- `stmt` `let x = 3`
- `block` `{ log(error, "hi"); return 12; }`
- `item` `fn foo() { }` `struct Bar`
- `meta` “” `cfg(target_os = "windows")`
- `tt`

- `expr` `=>` `,` `;`
- `ty` `path` `=>` `,` `:` `=` `>` `as`
- `pat` `=>` `,` `=`
-

Rust

`$($t:ty)* $e:expr` `$t` `$e` `$(T $t:ty)* E $e:exp`

/

`mod`

`fn`

`subsequent` `mod` `macro_use` `mod`

`macro_use` `extern crate`

```
#[macro_use(foo, bar)]
extern crate baz;
```

`#[macro_use]` `#[macro_use]` `#[macro_export]`

`#[no_link]`

```
macro_rules! m1 { () => (()) }

// visible here: m1

mod foo {
    // visible here: m1

    #[macro_export]
    macro_rules! m2 { () => (()) }

    // visible here: m1, m2
}

// visible here: m1

macro_rules! m3 { () => (()) }

// visible here: m1, m3

#[macro_use]
mod bar {
    // visible here: m1, m3

    macro_rules! m4 { () => (()) }

    // visible here: m1, m3, m4
}

// visible here: m1, m3, m4
```

`#[macro_use] extern crate` `m2`

Rust

`$crate`

`mylib`

```
pub fn increment(x: u32) -> u32 {
    x + 1
}

#[macro_export]
macro_rules! inc_a {
    ($x:expr) => ( ::increment($x) )
}

#[macro_export]
macro_rules! inc_b {
    ($x:expr) => ( ::mylib::increment($x) )
}
# fn main() { }
```

`inc_a` `mylib` `inc_b` `mylib` `inc_b`

Rust `foo` `$crate` `::foo` `$crate`

```
#[macro_export]
macro_rules! inc {
    ($x:expr) => ( $crate::increment($x) )
}
```

- `::increment` `::mylib::increment`
- `#[macro_use] extern crate ...` `mod` `$crate`

The deep end

Rust

```
macro_rules! bct {
    // cmd 0:  d ... => ...
    (0, $($ps:tt),* ; $_d:tt)
        => (bct!($($ps),*, 0 ; ));
    (0, $($ps:tt),* ; $_d:tt, $($ds:tt),*)
        => (bct!($($ps),*, 0 ; $($ds),*));

    // cmd 1p:  1 ... => 1 ... p
    (1, $p:tt, $($ps:tt),* ; 1)
        => (bct!($($ps),*, 1, $p ; 1, $p));
    (1, $p:tt, $($ps:tt),* ; 1, $($ds:tt),*)
        => (bct!($($ps),*, 1, $p ; 1, $($ds),*, $p));

    // cmd 1p:  0 ... => 0 ...
    (1, $p:tt, $($ps:tt),* ; $($ds:tt),*)
        => (bct!($($ps),*, 1, $p ; $($ds),*));

    // halt on empty data string
    ( $($ps:tt),* ; )
        => (());
}
```

`bct!`

Common macros

Rust

`panic!`

```
panic!("oh no!");
```

`vec!`

`vec!` `Vec<T>`

```
let v = vec![1, 2, 3, 4, 5];
```

vector100 `0`

```
let v = vec![0; 100];
```

`assert!` `assert_eq!`

`assert!` `assert_eq!` `panic!` Truth passes, success panic!s

```
// A-ok!

assert!(true);
assert_eq!(5, 3 + 2);

// nope :(

assert!(5 < 3);
assert_eq!(5, 3);
```

`try!`

`try!` `Result<T, E>` `T` `Ok<T>` `return` `Err(E)`

```
use std::fs::File;

fn foo() -> std::io::Result<()> {
    let f = try!(File::create("foo.txt"));

    Ok(())
}
```

```
use std::fs::File;

fn foo() -> std::io::Result<()> {
    let f = File::create("foo.txt");

    let f = match f {
        Ok(t) => t,
        Err(e) => return Err(e),
    };

    Ok(())
}
```

`unreachable!`

```
if false {
    unreachable!();
}
```

`panic!`

```
let x: Option<i32> = None;

match x {
    Some(_) => unreachable!(),
    None => println!("I know x is None!"),
}
```

`unimplemented!`

`unimplemented!` `unimplemented!`

Procedural macros

Rust `macro_rules!` bugRust *procedural macros*

---

1. libcollections `vec!`

> raw-pointers.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust Rust `unsafe`

`*const T` `*mut T` Rust“”Rust `*` `Rc<T>` `Arc<T>` Rust

- `Box` `&`
- `Box`
- `Box` Rustbug
- `*mut i32`
- `&`
- `*const T`

```rust
let x = 5;
let raw = &x as *const i32;

let mut y = 10;
let raw_mut = &mut y as *mut i32;
```

```rust
let x = 5;
let raw = &x as *const i32;

println!("raw points at {}", *raw);
```

```
error: dereference of unsafe pointer requires unsafe function or block [E0133]
     println!("raw points at{}", *raw);
                                 ^~~~
```

`unsafe`

```
let x = 5;
let raw = &x as *const i32;

let points_at = unsafe { *raw };

println!("raw points at {}", points_at);
```

API

FFI

FFIRust `*const T` `*mut T` C `const T*` `T*` FFI

`*` `&T` `*const T` `mut` `value as *const T` `value as *mut T`

`*const` `&` `&T` `*const T` `T`

```
// explicit cast
let i: u32 = 1;
let p_imm: *const u32 = &i as *const u32;

// implicit coercion
let mut m: u32 = 2;
let p_mut: *mut u32 = &mut m;

unsafe {
    let ref_imm: &u32 = &*p_imm;
    let ref_mut: &mut u32 = &mut *p_mut;
}
```

`transmute` `&*x` `x` `transmute`

> unsafe.md
> commit 07aaca3a0724000e735a558d4c23b600512346d9

RustRust `unsafe` `unsafe`

`unsafe`

```
unsafe fn danger_will_robinson() {
    // scary stuff
}
```

FFI `unsafe` `unsafe`

```
unsafe {
    // scary stuff
}
```

trait

```
unsafe trait Scary { }
```

`impl` trait

```
# unsafe trait Scary { }
unsafe impl Scary for i32 {}
```

bugRust `unsafe`

“”What does ‘safe’ mean?

Rust“”

-
-
-
-

RustbugRust `unsafe`

Rust `unsafe`

-
- /
- `undef`
-

- `&mut T` `&T` LLVM [`noalias`] `&T` `UnsafeCell<U>`
- `UnsafeCell<U>` /
-
    - `std::ptr::offset` `offset`
    - `std::ptr::copy_nonoverlapping_memory` `memcpy32/memcpy64`
- /
    - /
    - `bool` `false` `0` `true` `1`
    - `enum`
    - `char` `char::MAX`
    - `str` UTF-8
- RustRust

Unsafe Superpowers

Rust33

1.
2.
3. NB

> `unsafe` “”Rust `unsafe`

3

`static mut`

Rust `static mut`

`unsafe` `unsafe`

Rust

> The compiler will act as though you’re upholding its invariants

Rust

> effective-rust.md commit f01dbf21945aa4d1a11d0ba1695238c59bdf4a44

Rust Rust Rust

Rust

> the-stack-and-the-heap.md
> commit 049b9e4e8067b998e4581d026b0bc6d1113ab9f5

Rust C Rust

Rust

Rust

```rust
fn main() {
    let x = 42;
}
```

`x` Rust“”“”

“stack frame” `main()` 32Rust

“”

```rust
fn foo() {
    let y = 5;
    let z = 100;
}

fn main() {
    let x = 42;

    foo();
}
```

3 `foo()` `main()` `main()` `foo()` `0` 1GB `0` `1,073,741,824` 2 ^{30}1GB gigabyte

> [gigabyte]: `Gigabyte` 10^9 2^30SI“gigabyte” 10^9 “gibibyte” 2^30

`0`

0	x	42

`0` `x` `42`

`foo()`

2	z	100
1	y	5
0	x	42

`0` `1` `2` `foo()`

`0` `1` `2`

`foo()`

0	x	42

`main()`

“”“”

```rust
fn italic() {
    let i = 6;
}

fn bold() {
    let a = 5;
    let b = 100;
    let c = 1;

    italic();
}

fn main() {
    let x = 42;

    bold();
}
```

`main()`

0	x	42

`main()` `bold()`

3	**c**	**1**
2	**b**	**100**
1	**a**	**5**
0	x	42

`bold()` `italic()`

4	*i*	*6*
3	**c**	**1**
2	**b**	**100**
1	**a**	**5**
0	x	42

`italic()` `bold()` `main()`

3	**c**	**1**
2	**b**	**100**
1	**a**	**5**
0	x	42

`bold()` `main()`

0	x	42

Rust `Box<T>`

```rust
fn main() {
    let x = Box::new(5);
    let y = 42;
}
```

`main()`

1	y	42
0	x	??????

`y` `42` `x` `x` `Box<i32>` “” `Box::new()` `5`

$(2^{30}) - 1$		5
...	...	...
1	y	42
0	x	→ $(2^{30}) - 1$

1GB RAM $(2^{30}) - 1$ `0` `x` `x` $(2^{30}) - 1$

“”

$(2^{30}) - 1$		5
$(2^{30}) - 2$		
$(2^{30}) - 3$		
$(2^{30}) - 4$		42
...	...	...
3	y	→ $(2^{30}) - 4$
2	y	42
1	y	42
0	x	→ $(2^{30}) - 1$

4 $(2^{30}) - 1$ $(2^{30}) - 4$ gap“”Rust jemalloc

[] `main()` `Box<T>` Drop `Box` `Drop` `x`

1	y	42
0	x	??????

[]“”

Rust

```
fn foo(i: &i32) {
    let z = 42;
}

fn main() {
    let x = 5;
    let y = &x;

    foo(y);
}
```

`main()`

1	y	→ 0
0	x	5

`x 5 y x` `x` `0`

`foo() y`

3	z	42
2	i	→ 0
1	y	→ 0
0	x	5

`i` `z` `i y` `y 0 i`

```
 fn foo(x: &i32) {
    let y = 10;
    let z = &y;

    baz(z);
    bar(x, z);
}

fn bar(a: &i32, b: &i32) {
    let c = 5;
    let d = Box::new(5);
    let e = &d;

    baz(e);
}

fn baz(f: &i32) {
    let g = 100;
}

fn main() {
    let h = 3;
    let i = Box::new(20);
    let j = &h;

    foo(j);
}
```

`main()`

$(2^{30}) - 1$		20
...	...	...
2	j	$\rightarrow$ 0
1	i	$\rightarrow (2^{30}) - 1$
0	h	3

`j` `i` `h` `i`

`main()` `foo()`

$(2^{30}) - 1$		20
...	...	...
5	z	→ 4
4	y	10
3	x	→ 0
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

`x y z` `x j` `0` `j h`

`foo() baz() z`

$(2^{30}) - 1$		20
...	...	...
7	g	100
6	f	→ 4
5	z	→ 4
4	y	10
3	x	→ 0
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

`f g` `baz()`

$(2^{30}) - 1$		20
...	...	...
5	z	→ 4
4	y	10
3	x	→ 0
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

`foo()` `bar()` `x` `z`

$(2^{30}) - 1$		20
$(2^{30}) - 2$		5
...	...	...
10	e	→ 9
9	d	→ $(2^{30}) - 2$
8	c	5
7	b	→ 4
6	a	→ 0
5	z	→ 4
4	y	10
3	x	→ 0
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

$(2^{30}) - 1$ `1,073,741,822`

`bar()` `baz()`

$(2^{30}) - 1$		20
$(2^{30}) - 2$		5
...	...	...
12	g	100
11	f	→ $(2^{30}) - 2$
10	e	→ 9
9	d	→ $(2^{30}) - 2$
8	c	5
7	b	→ 4
6	a	→ 0
5	z	→ 4
4	y	10
3	x	→ 0
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

`baz()` `f` `g`

$(2^{30}) - 1$		20
$(2^{30}) - 2$		5
...	...	...
10	e	→ 9
9	d	→ $(2^{30}) - 2$
8	c	5
7	b	→ 4
6	a	→ 0
5	z	→ 4
4	y	10
3	x	→ 0
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

`bar()` `d` `Box<T>` $(2^{30}) - 1$

$(2^{30}) - 1$		20
...	...	...
5	z	→ 4
4	y	10
3	x	→ 0
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

`foo()`

$(2^{30}) - 1$		20
...	...	...
2	j	→ 0
1	i	→ $(2^{30}) - 1$
0	h	3

`main()` `i` `Drop`

100% `Drop`

Which to use?

Rust

trivialC++“”non-trivial

Semantic impact

Rust mental model Rust `Rc<T>` `Arc<T>`

Rust

> testing.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4
>
> Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.
>
> Edsger W. Dijkstra, "The Humble Programmer" (1972)
>
> bug
>
> Edsger W. Dijkstra1972

RustRust

`test` The test attribute

`test` Cargo `adder`

```
$ cargo new adder
$ cd adder
```

Cargo `src/lib.rs`

```
#[test]
fn it_works() {
}
```

`#[test]` `cargo test`

```
$ cargo test
   Compiling adder v0.0.1 (file:///home/you/projects/adder)
     Running target/adder-91b3e234d4ed382a

running 1 test
test it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

   Doc-tests adder

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured
```

Cargo

```
test it_works ... ok
```

`it_works`

```
fn it_works() {
```

```
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured
```

`panic!` `panic!`

```
#[test]
fn it_works() {
    assert!(false);
}
```

`assert!` Rust `true` `false` `panic!`

```
$ cargo test
   Compiling adder v0.0.1 (file:///home/you/projects/adder)
     Running target/adder-91b3e234d4ed382a

running 1 test
test it_works ... FAILED

failures:

---- it_works stdout ----
        thread 'it_works' panicked at 'assertion failed: false', /home/steve/tmp/adder/
src/lib.rs:3



failures:
    it_works

test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured

thread '<main>' panicked at 'Some tests failed', /home/steve/src/rust/src/libtest/lib.r
s:247
```

Rust

```
test it_works ... FAILED
test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured
```

0. OS X Linux `$?`

```
$ echo $?
101
```

Windows `cmd`

```
> echo %ERRORLEVEL%
```

PowerShell

```
> echo $LASTEXITCODE # the code itself
> echo $? # a boolean, fail or succeed
```

`cargo test`

`should_panic`

```
#[test]
#[should_panic]
fn it_works() {
    assert!(false);
}
```

`panic!`

```
$ cargo test
   Compiling adder v0.0.1 (file:///home/you/projects/adder)
     Running target/adder-91b3e234d4ed382a

running 1 test
test it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

   Doc-tests adder

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured
```

Rust `assert_eq!`

```
#[test]
#[should_panic]
fn it_works() {
    assert_eq!("Hello", "world");
}
```

`should_panic`

```
$ cargo test
   Compiling adder v0.0.1 (file:///home/you/projects/adder)
     Running target/adder-91b3e234d4ed382a

running 1 test
test it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

   Doc-tests adder

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured
```

`should_panic` `should_panic` `expected`

```
#[test]
#[should_panic(expected = "assertion failed")]
fn it_works() {
    assert_eq!("Hello", "world");
}
```

“”

```
pub fn add_two(a: i32) -> i32 {
    a + 2
}

#[test]
fn it_works() {
    assert_eq!(4, add_two(2));
}
```

`assert_eq!`

`ignore`

`ignore`

```
#[test]
fn it_works() {
    assert_eq!(4, add_two(2));
}

#[test]
#[ignore]
fn expensive_test() {
    // code that takes an hour to run
}
```

`it_works` `expensive_test`

```
$ cargo test
   Compiling adder v0.0.1 (file:///home/you/projects/adder)
     Running target/adder-91b3e234d4ed382a

running 2 tests
test expensive_test ... ignored
test it_works ... ok

test result: ok. 1 passed; 0 failed; 1 ignored; 0 measured

   Doc-tests adder

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured
```

`cargo test -- --ignored`

```
$ cargo test -- --ignored
     Running target/adder-91b3e234d4ed382a

running 1 test
test expensive_test ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

   Doc-tests adder

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured
```

`--ignored` test Cargo `cargo test -- --ignored`

`tests`

`tests`

```rust
pub fn add_two(a: i32) -> i32 {
    a + 2
}

#[cfg(test)]
mod tests {
    use super::add_two;

    #[test]
    fn it_works() {
        assert_eq!(4, add_two(2));
    }
}
```

`cfg` `mod tests` `cfg`

`use` `glob` `src/lib.rs`

```rust
pub fn add_two(a: i32) -> i32 {
    a + 2
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn it_works() {
        assert_eq!(4, add_two(2));
    }
}
```

`use`

```
$ cargo test
    Updating registry `https://github.com/rust-lang/crates.io-index`
   Compiling adder v0.0.1 (file:///home/you/projects/adder)
     Running target/adder-91b3e234d4ed382a

running 1 test
test test::it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

   Doc-tests adder

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured
```

`test` “”“” `tests`

`tests`

`tests` `tests/lib.rs`

```
extern crate adder;

#[test]
fn it_works() {
    assert_eq!(4, adder::add_two(2));
}
```

`extern crate adder` `tests` `tests`

```
$ cargo test
   Compiling adder v0.0.1 (file:///home/you/projects/adder)
     Running target/adder-91b3e234d4ed382a

running 1 test
test test::it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

     Running target/lib-c18e7d3494509e74

running 1 test
test it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

   Doc-tests adder

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured
```

`tests` `test`

Rust crate crate `src/lib.rs`

```
//! The `adder` crate provides functions that add numbers to other numbers.
//!
//! # Examples
//!
//! ` ` ` gitbook
//! assert_eq!(4, adder::add_two(2));
//! ` ` ` gitbook

/// This function adds two to its argument.
///
/// # Examples
///
/// ` ` ` gitbook
/// use adder::add_two;
///
/// assert_eq!(4, add_two(2));
/// ` ` ` gitbook
pub fn add_two(a: i32) -> i32 {
    a + 2
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn it_works() {
        assert_eq!(4, add_two(2));
    }
}
```

`//!` `///` RustMarkdown3 `# Examples`

```
$ cargo test
   Compiling adder v0.0.1 (file:///home/steve/tmp/adder)
     Running target/adder-91b3e234d4ed382a

running 1 test
test test::it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

     Running target/lib-c18e7d3494509e74

running 1 test
test it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured

   Doc-tests adder

running 2 tests
test add_two_0 ... ok
test _0 ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured
```

3 `_0` `add_two_0` `add_two_1`

crate crate

> conditional-compilation.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust `#[cfg]`

```
#[cfg(foo)]
# fn foo() {}

#[cfg(bar = "baz")]
# fn bar() {}
```

```
#[cfg(any(unix, windows))]
# fn foo() {}

#[cfg(all(unix, target_pointer_width = "32"))]
# fn bar() {}

#[cfg(not(foo))]
# fn not_foo() {}
```

```
#[cfg(any(not(unix), all(target_os="macos", target_arch = "powerpc")))]
# fn foo() {}
```

Cargo `Cargo.toml [features]`

```
[features]
# no features by default
default = []

# The “secure-password” feature depends on the bcrypt package.
secure-password = ["bcrypt"]
```

Cargo `rustc`

```
--cfg feature="${feature_name}"
```

`cfg`

```
#[cfg(feature = "foo")]
mod foo {
}
```

`cargo build --features "foo"` `rustc --cfg feature="foo"` `mod foo` `cargo build`
`foo`

cfg_attr

`cfg` `cfg_attr`

```
#[cfg_attr(a, b)]
# fn foo() {}
```

`a` `cfg` `#[b]`

cfg!

`cfg!`

```
if cfg!(target_os = "macos") || cfg!(target_os = "ios") {
    println!("Think Different!");
}
```

`true` `false`

}

> documentation.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4
>
> gitbookmarkdown3githubbug

RustRust

`rustdoc`

Rust `rustdoc` `rustdoc` Cargo `cargo doc`

Markdown

Rust

```
/// Constructs a new `Rc<T>`.
///
/// # Examples
///
/// ` ` `
/// use std::rc::Rc;
///
/// let five = Rc::new(5);
/// ` ` `
pub fn new(value: T) -> Rc<T> {
    // implementation goes here
}
```

`///` `//`

Markdown

Rust

```
/// The `Option` type. See [the module level documentation](../) for more.
enum Option<T> {
    /// No value
    None,
    /// Some value `T`
    Some(T),
}
```

```
/// The `Option` type. See [the module level documentation](../) for more.
enum Option<T> {
    None, /// No value
    Some(T), /// Some value `T`
}
```

```
hello.rs:4:1: 4:2 error: expected ident, found `}`
hello.rs:4 }
           ^
```

```
/// Constructs a new `Rc<T>`.
# fn foo() {}
```

```
///
/// Other details about constructing `Rc<T>`s, maybe describing complicated
/// semantics, maybe additional options, all kinds of stuff.
///
# fn foo() {}
```

`#`

```
/// # Panics
# fn foo() {}
```

Rust

```
/// # Failures
# fn foo() {}
```

`Result<T, E>` `Err(E)` `Panics`

```
/// # Safety
# fn foo() {}
```

`unsafe`

```
/// # Examples
///
/// ` ` `
/// use std::rc::Rc;
///
/// let five = Rc::new(5);
/// ` ` `
# fn foo() {}
```

`Examples` aishang

```
/// # Examples
///
/// Simple `&str` patterns:
///
/// ` ` `
/// let v: Vec<&str> = "Mary had a little lamb".split(' ').collect();
/// assert_eq!(v, vec!["Mary", "had", "a", "little", "lamb"]);
/// ` ` `
///
/// More complex patterns with a lambda:
///
/// ` ` `
/// let v: Vec<&str> = "abc1def2ghi".split(|c: char| c.is_numeric()).collect();
/// assert_eq!(v, vec!["abc", "def", "ghi"]);
/// ` ` `
# fn foo() {}
```

Rust

```
/// ` ` `
/// println!("Hello, world");
/// ` ` `
# fn foo() {}
```

Rust

```
/// ` ` `c
/// printf("Hello, world\n");
/// ` ` `
# fn foo() {}
```

`text`

`rustdoc` C `rustdoc` Rust `rustdoc`

```
/// ` ` `
/// println!("Hello, world");
/// ` ` `
# fn foo() {}
```

`fn main()` `rustdoc main()`

```
/// ` ` `
/// use std::rc::Rc;
///
/// let five = Rc::new(5);
/// ` ` `
# fn foo() {}
```

```
fn main() {
    use std::rc::Rc;
    let five = Rc::new(5);
}
```

`rustdoc`

1. `#![foo]`
2. `allow` `unused_variables` `unused_assignments` `unused_mut` `unused_attributes` `dead_code` lint
3. `extern crate` `extern crate <mycrate>;`
4. `fn main` `fn main() { your_code }`

`///`

```
/// Some documentation.
# fn foo() {}
```

```
/// Some documentation.
# fn foo() {}
```

`#`

```
let x = 5;
let y = 6;
println!("{}", x + y);
```

> `x` `5`
>
> ```
> let x = 5;
> # let y = 6;
> # println!("{}", x + y);
> ```
>
> `y` `6`
>
> ```
> # let x = 5;
> let y = 6;
> # println!("{}", x + y);
> ```
>
> `x` `y`
>
> ```
> # let x = 5;
> # let y = 6;
> println!("{}", x + y);
> ```

~~~
    `x``5`

    ```text
    let x = 5;
    # let y = 6;
    # println!("{}", x + y);
~~~

~~~
`y``6`

```text
# let x = 5;
let y = 6;
# println!("{}", x + y);
```

`x``y`

```text
# let x = 5;
# let y = 6;
println!("{}", x + y);
```
~~~

~~~

###


```rust
/// Panic with a given message unless an expression evaluates to true.
///
/// # Examples
///
/// ` ` `
/// # #[macro_use] extern crate foo;
/// # fn main() {
/// panic_unless!(1 + 1 == 2, “Math is broken.”);
/// # }
/// ` ` `
///
/// ` ` `should_panic
/// # #[macro_use] extern crate foo;
/// # fn main() {
/// panic_unless!(true == false, “I’m broken.”);
/// # }
/// ` ` `
#[macro_export]
macro_rules! panic_unless {
    ($condition:expr, $($rest:expr),+) => ({ if ! $condition { panic!($($rest),+); } })
;
}
~~~

3 `extern crate` `#[macro_use]` `main()` `#`

`#`

```
/// use std::io;
/// let mut input = String::new();
/// try!(io::stdin().read_line(&mut input));
```

`try!` `Result<T, E>`

```
/// A doc test using try!
///
/// ` ` `
/// use std::io;
/// # fn foo() -> io::Result<()> {
/// let mut input = String::new();
/// try!(io::stdin().read_line(&mut input));
/// # Ok(())
/// # }
/// ` ` `
# fn foo() {}
```

`Result<T, E>`

```
$ rustdoc --test path/to/my/crate/root.rs
# or
$ cargo test
```

`cargo test` **`cargo test` crate crate** `rustdoc`

`rustdoc`

```
/// ` ` `ignore
/// fn foo() {
/// ` ` `
# fn foo() {}
```

`ignore` Rust `text` `#`

```
/// ` ` `should_panic
/// assert!(false);
/// ` ` `
# fn foo() {}
```

`should_panic` `rustdoc`

```
/// ` ` `no_run
/// loop {
///     println!("Hello, world");
/// }
/// ` ` `
# fn foo() {}
```

`no_run` “”

Rust `//!`

```
mod foo {
    //! This is documentation for the `foo` module.
    //!
    //! # Examples

    // ...
}
```

`//!` `foo.rs`

```
//! A module for using `foo`s.
//!
//! The `foo` module contains a lot of useful functionality blah blah blah
```

RFC 505

Rust Markdown `.md`

Markdown

```
/// # Examples
///
/// ` ` `
/// use std::rc::Rc;
///
/// let five = Rc::new(5);
/// ` ` `
```

> ```
> # Examples
>
> use std::rc::Rc;
>
> let five = Rc::new(5);
> ```

MarkdownMarkdown

```
% The title

This is the example documentation.
```

`%`

`doc`

```
/// this

#[doc="this"]
```

```
//! this

#![doc="/// this"]
```

Re-exports

`rustdoc`

```
extern crate foo;

pub use foo::bar;
```

`foo`

`no_inline`

```
extern crate foo;

#[doc(no_inline)]
pub use foo::bar;
```

Rust `warn`

```
#![warn(missing_docs)]
```

`deny`

```
#![deny(missing_docs)]
```

/ `allow`

```
#[allow(missing_docs)]
struct Undocumented;
```

```
#[doc(hidden)]
struct Hidden;
```

HTML

`#![doc]` `rustdoc` THML

```
#![doc(html_logo_url = "http://www.rust-lang.org/logos/rust-logo-128x128-blk-v2.png",
       html_favicon_url = "http://www.rust-lang.org/favicon.ico",
       html_root_url = "http://doc.rust-lang.org/")];
```

logoURL

`rustdoc`

- `--html-in-header FILE` `<head>...</head>` `FILE`
- `--html-before-content FILE` `<body>` `FILE`
- `--html-after-content FILE` `FILE`

MarkdownHTMLXSS

```
/// <script>alert(document.cookie)</script>
# fn foo() {}
```

> iterators.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust `for`

```
for x in 0..10 {
    println!("{}", x);
}
```

Rust `0..10` “” `.next()`

```
let mut range = 0..10;

loop {
    match range.next() {
        Some(x) => {
            println!("{}", x);
        },
        None => { break }
    }
}
```

`range` `loop` `match` `match` `range.next()` `next` `Option<i32>` , `Some(i32)` , `None` `Some(i32)` `None` `break`

`loop` `for` `loop` / `match` / `break`

`for` `Iterator` RustRust

```
let nums = vec![1, 2, 3];

for i in 0..nums.len() {
    println!("{}", nums[i]);
}
```

```rust
let nums = vec![1, 2, 3];

for num in &nums {
    println!("{}", num);
}
```

`nums[i]`

100%`println!` `num` `&i32` `i32` `i32` `println!`

```rust
let nums = vec![1, 2, 3];

for num in &nums {
    println!("{}", *num);
}
```

`num` `&nums` `&`

3 *iterator adapters**consumers*

-
-
-

Consumers

`collect()`

```rust
let one_to_one_hundred = (1..101).collect();
```

`collect()` `collect()` Rust

```rust
let one_to_one_hundred = (1..101).collect::<Vec<i32>>();
```

`::<>` `_`

```rust
let one_to_one_hundred = (1..101).collect::<Vec<_>>();
```

“ `Vec<T>` `T` ” `_` “”

`collect()` `find()`

```
let greater_than_forty_two = (0..100)
                             .find(|x| *x > 42);

match greater_than_forty_two {
    Some(_) => println!("We got some numbers!"),
    None => println!("No numbers found :("),
}
```

`find` `true` `false` `find` `Option`

`fold`

```
let sum = (1..4).fold(0, |sum, x| sum + x);
```

`fold()` `fold(base, |accumulator, element| ...)` *base* *accumulator* *element*

0	0	1	1
0	1	2	3
0	3	3	6

`fold()`

```
# (1..4)
.fold(0, |sum, x| sum + x);
```

`0` `sum` `x` `sum` `0` `x` `nums` `1` `sum` `x` `0 + 1 = 1` `sum` `2` `1 + 2 = 3` `x` `3` `3 + 3 = 6` `1 + 2 + 3 = 6`

`fold` `fold`

Iterators

`.next()` *lazy* `1-99`

```
let nums = 1..100;
```

```
let nums = (1..100).collect::<Vec<i32>>();
```

`collect()`

`iter()` `iter()`

```
let nums = [1, 2, 3];

for num in nums.iter() {
   println!("{}", num);
}
```

Iterator adapters

Iterator adapters `map`

```
(1..100).map(|x| x + 1);
```

`map` `2-100`

```
warning: unused result which must be used: iterator adaptors are lazy and
         do nothing unless consumed, #[warn(unused_must_use)] on by default
(1..100).map(|x| x + 1);
 ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
(1..100).map(|x| println!("{}", x));
```

`for`

`take(n)` `n` `count()`

```
for i in (1..).take(5) {
    println!("{}", i);
}
```

```
1
2
3
4
5
```

`filter()` `true` `false` `filter()` `true`

```
for i in (1..100).filter(|&x| x % 2 == 0) {
    println!("{}", i);
}
```

1100 `filter` `&x`

```
(1..)
    .filter(|&x| x % 2 == 0)
    .filter(|&x| x % 3 == 0)
    .take(5)
    .collect::<Vec<i32>>();
```

`6 12 18 24 30`

> concurrency.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust Rust Rust

Rust Rust Rust mio

`Send` `Sync`

Rust Rust

`Send`

Send `T` `Send`

`Send`

FFI `Send`

`Sync`

`Sync` `T` `Sync` `Sync`

Rust `Arc<T>` wrapper `Arc<T>` `Send` `Sync` `T` `Send` `Sync` `Arc<RefCell<U>>` RefCell `Sync` `Arc<RefCell<U>>` `Send`

Rust `Arc<T>` `Sync`

Rust

Rust“” Rust `std::thread`

```
use std::thread;

fn main() {
    thread::spawn(|| {
        println!("Hello from a thread!");
    });
}
```

`thread::spawn()`

```
use std::thread;

fn main() {
    let handle = thread::spawn(|| {
        "Hello from a thread!"
    });

    println!("{}", handle.join().unwrap());
}
```

Rust

Safe Shared Mutable State

Rust“”

> “”Rust“”

bug

Rust

```
use std::thread;
use std::time::Duration;

fn main() {
    let mut data = vec![1, 2, 3];

    for i in 0..3 {
        thread::spawn(move || {
            data[i] += 1;
        });
    }

    thread::sleep(Duration::from_millis(50));
}
```

```
8:17 error: capture of moved value: `data`
        data[i] += 1;
        ^~~~
```

Rust `data` 3

`Sync`

`Arc<T>` Rust

`Arc<T>`

```
use std::thread;
use std::sync::Arc;
use std::time::Duration;

fn main() {
    let mut data = Arc::new(vec![1, 2, 3]);

    for i in 0..3 {
        let data = data.clone();
        thread::spawn(move || {
            data[i] += 1;
        });
    }

    thread::sleep(Duration::from_millis(50));
}
```

`Arc<T>` `clone()`

```
:11:24 error: cannot borrow immutable borrowed content as mutable
:11                    data[i] += 1;
                             ^~~~
```

`Arc<T>` `Sync`

`Mutex<T>`

```
use std::sync::{Arc, Mutex};
use std::thread;
use std::time::Duration;

fn main() {
    let data = Arc::new(Mutex::new(vec![1, 2, 3]));

    for i in 0..3 {
        let data = data.clone();
        thread::spawn(move || {
            let mut data = data.lock().unwrap();
            data[i] += 1;
        });
    }

    thread::sleep(Duration::from_millis(50));
}
```

`i`

Mutexlock

```
fn lock(&self) -> LockResult<MutexGuard<T>>
```

`MutexGuard<T>` `Send` guard

```
# use std::sync::{Arc, Mutex};
# use std::thread;
# use std::time::Duration;
# fn main() {
#     let data = Arc::new(Mutex::new(vec![1, 2, 3]));
#     for i in 0..3 {
#         let data = data.clone();
thread::spawn(move || {
    let mut data = data.lock().unwrap();
    data[i] += 1;
});
#     }
#     thread::sleep(Duration::from_millis(50));
# }
```

`lock()` `Result<T, E>` `unwrap()`

Rust

Channels

```rust
use std::sync::{Arc, Mutex};
use std::thread;
use std::sync::mpsc;

fn main() {
    let data = Arc::new(Mutex::new(0));

    let (tx, rx) = mpsc::channel();

    for _ in 0..10 {
        let (data, tx) = (data.clone(), tx.clone());

        thread::spawn(move || {
            let mut data = data.lock().unwrap();
            *data += 1;

            tx.send(()).unwrap();
        });
    }

    for _ in 0..10 {
        rx.recv().unwrap();
    }
}
```

`mpsc::channel()` `send ()` 10

`Send`

```rust
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();

    for i in 0..10 {
        let tx = tx.clone();

        thread::spawn(move || {
            let answer = i * i;

            tx.send(answer).unwrap();
        });
    }

    for _ in 0..10 {
        println!("{}", rx.recv().unwrap());
    }
}
```

10 `spawn()` i `send()`

Panics

`panic!` Rust

```
use std::thread;

let handle = thread::spawn(move || {
    panic!("oops!");
});

let result = handle.join();

assert!(result.is_err());
```

`Thread` `Result` ,

> error-handling.md
> commit e26279db48cc5510a13f0e97bde97ccd2d2a1854

Rust Rust

Rust

Rust

Rust

-
  - unwrapping
  - `Option`
    - `Option<T>`
  - `Result`
    -
    - `Result`
  - unwrapping
-
  - `Option` `Result`
  -
  -
  - `try!`
  -
- trait
  - `Error` trait
  - `From` trait
  - `try!` macro
  -
  -
-
  -
  -
  -
  - `Box<Error>`
  -
  -
  -
-

case analysis

panic `panic`

```rust
// Guess a number between 1 and 10.
// If it matches the number we had in mind, return true. Else, return false.
fn guess(n: i32) -> bool {
    if n < 1 || n > 10 {
        panic!("Invalid number: {}", n);
    }
    n == 5
}

fn main() {
    guess(11);
}
```

```
thread '
' panicked at 'Invalid number: 11', src/bin/panic-simple.rs:5
```

```rust
use std::env;

fn main() {
    let mut argv = env::args();
    let arg: String = argv.nth(1).unwrap(); // error 1
    let n: i32 = arg.parse().unwrap(); // error 2
    println!("{}", 2 * n);
}
```

0 1 2 panic

unwrapping

panic `panic` panic `unwrap`

Rust “unwrap”“panic ” unwrap `Option` `Result` `unwrap`

`Option`

`Option`

```rust
enum Option<T> {
    None,
    Some(T),
}
```

`Option` Rust *possibility of absence*

```rust
// Searches `haystack` for the Unicode character `needle`. If one is found, the
// byte offset of the character is returned. Otherwise, `None` is returned.
fn find(haystack: &str, needle: char) -> Option<usize> {
    for (offset, c) in haystack.char_indices() {
        if c == needle {
            return Some(offset);
        }
    }
    None
}
```

`offset` `Some(offset)` `Some` `Option` `fn<T>(value: T) -> Option<T>` `None` `None` `fn<T>() -> Option<T>`

`find`

```rust
# fn find(_: &str, _: char) -> Option<usize> { None }
fn main() {
    let file_name = "foobar.rs";
    match find(file_name, '.') {
        None => println!("No file extension found."),
        Some(i) => println!("File extension: {}", &file_name[i+1..]),
    }
}
```

`find` `Option<usize>` case analysiscase analysis `Option<T>` `Option<T>` `None` `Some(t)`

`unwrap` case analysiscase analysis `unwrap`

```
enum Option<T> {
    None,
    Some(T),
}

impl<T> Option<T> {
    fn unwrap(self) -> T {
        match self {
            Option::Some(val) => val,
            Option::None =>
              panic!("called `Option::unwrap()` on a `None` value"),
        }
    }
}
```

`unwrap` case analysis `unwrap` `panic!` `unwrap`

`Option<T>`

`find` `.` `Option<T>`

```
# fn find(_: &str, _: char) -> Option<usize> { None }
// Returns the extension of the given file name, where the extension is defined
// as all characters proceeding the first `.`.
// If `file_name` has no `.`, then `None` is returned.
fn extension_explicit(file_name: &str) -> Option<&str> {
    match find(file_name, '.') {
        None => None,
        Some(i) => Some(&file_name[i+1..]),
    }
}
```

extension

`find` `extension_explicit` case analysis

`extension_explicit` case analysis `Option<T>` `None` `None`

Rust parametric polymorphism

```
fn map<F, T, A>(option: Option<T>, f: F) -> Option<A> where F: FnOnce(T) -> A {
    match option {
        None => None,
        Some(value) => Some(f(value)),
    }
}
```

`map` `Option<T>`

`extension_explicit` case analysis

```rust
# fn find(_: &str, _: char) -> Option<usize> { None }
// Returns the extension of the given file name, where the extension is defined
// as all characters proceeding the first `.`.
// If `file_name` has no `.`, then `None` is returned.
fn extension(file_name: &str) -> Option<&str> {
    find(file_name, '.').map(|i| &file_name[i+1..])
}
```

`Option` `None` `rs` case analysis - `Option<T>`

```rust
fn unwrap_or<T>(option: Option<T>, default: T) -> T {
    match option {
        None => default,
        Some(value) => value,
    }
}
```

`Option<T>`

```rust
# fn find(haystack: &str, needle: char) -> Option<usize> {
#     for (offset, c) in haystack.char_indices() {
#         if c == needle {
#             return Some(offset);
#         }
#     }
#     None
# }
#
# fn extension(file_name: &str) -> Option<&str> {
#     find(file_name, '.').map(|i| &file_name[i+1..])
# }
fn main() {
    assert_eq!(extension("foobar.csv").unwrap_or("rs"), "csv");
    assert_eq!(extension("foobar").unwrap_or("rs"), "rs");
}
```

`unwrap_or` `Option<T>` `unwrap_or_else`

combinator `and_then` `. .. /`

case analysis

```
# fn extension(file_name: &str) -> Option<&str> { None }
fn file_path_ext_explicit(file_path: &str) -> Option<&str> {
    match file_name(file_path) {
        None => None,
        Some(name) => match extension(name) {
            None => None,
            Some(ext) => Some(ext),
        }
    }
}

fn file_name(file_path: &str) -> Option<&str> {
  // implementation elided
  unimplemented!()
}
```

`map` case analysis `map` Option `Some` `map Option` `map`

```
fn and_then<F, T, A>(option: Option<T>, f: F) -> Option<A>
        where F: FnOnce(T) -> Option<A> {
    match option {
        None => None,
        Some(value) => f(value),
    }
}
```

case analysis `file_path_ext`

```
# fn extension(file_name: &str) -> Option<&str> { None }
# fn file_name(file_path: &str) -> Option<&str> { None }
fn file_path_ext(file_path: &str) -> Option<&str> {
    file_name(file_path).and_then(extension)
}
```

`Option` —— case analysis `Result` `Result`

`Option` case analysis `unwrap Option<T> None` panic

`Result`

`Result`

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

`Result Option` `Option` `Result` `Option` `Option<T>`

```
type Option<T> = Result<T, ()>;
```

`Result` `()` “”“” `()` `()`

`Result` “ `Ok` ”“ `Err` ”

`Option` `Result` `unwrap`

```
# enum Result<T, E> { Ok(T), Err(E) }
impl<T, E: ::std::fmt::Debug> Result<T, E> {
    fn unwrap(self) -> T {
        match self {
            Result::Ok(val) => val,
            Result::Err(err) =>
              panic!("called `Result::unwrap()` on an `Err` value: {:?}", err),
        }
    }
}
```

`Option::unwrap` `panic!` `E` `Debug` `Debug` `Debug`

OK

Rust

```
fn double_number(number_str: &str) -> i32 {
    2 * number_str.parse::<i32>().unwrap()
}

fn main() {
    let n: i32 = double_number("10");
    assert_eq!(n, 20);
}
```

`unwrap` panic

```
thread '
' panicked at 'called `Result::unwrap()` on an `Err` value: ParseIntError { kind: Inval
idDigit }', /home/rustbuild/src/rust-buildbot/slave/beta-dist-rustc-linux/build/src/lib
core/result.rs:729
```

`double_number` `parse`

```
impl str {
    fn parse<F: FromStr>(&self) -> Result<F, F::Err>;
}
```

`Result` `Option` `Result` “”
`Opation` `Result`

`parse` `i32` `FromStr` `CTRL-F` “FromStr” ` `Err`
`std::num::ParseIntError`

```
use std::num::ParseIntError;

fn double_number(number_str: &str) -> Result<i32, ParseIntError> {
    match number_str.parse::<i32>() {
        Ok(n) => Ok(2 * n),
        Err(err) => Err(err),
    }
}

fn main() {
    match double_number("10") {
        Ok(n) => assert_eq!(n, 20),
        Err(err) => println!("Error: {:?}", err),
    }
}
```

case analysis

`Opation` `Result` `Result` `option` `map`

```
use std::num::ParseIntError;

fn double_number(number_str: &str) -> Result<i32, ParseIntError> {
    number_str.parse::<i32>().map(|n| 2 * n)
}

fn main() {
    match double_number("10") {
        Ok(n) => assert_eq!(n, 20),
        Err(err) => println!("Error: {:?}", err),
    }
}
```

`Result` unwrap_orand_then `Result` map_err `map` or_else `and_then`

`Result`

`Result<i32>` `Result` `Resule`

```
use std::num::ParseIntError;
use std::result;

type Result<T> = result::Result<T, ParseIntError>;

fn double_number(number_str: &str) -> Result<i32> {
    unimplemented!();
}
```

`ParseIntError` `ParseIntError`

`io::Result` `io::Result<T>` `io` `std::result` `fmt::Result`

unwrapping

`unwrap` `panic`

`unwrap` `unwrap`

- `unwrap`
- **panic bug** panic bug `assert!`

`Option` `expect` `expect` `unwrap` `expect` panic “called unwrap on a None value.”

“ X”“Y ”

Rust unwrap

`Option<T>` `Result<T, SomeError>` `Option` `Result` `Result<T, Error1>` `Result<T, Error2>`

`Option` `Result`

`Option` `Result` case analysis

`Option` `Result` case analysis

```
use std::env;

fn main() {
    let mut argv = env::args();
    let arg: String = argv.nth(1).unwrap(); // error 1
    let n: i32 = arg.parse().unwrap(); // error 2
    println!("{}", 2 * n);
}
```

`Opation  Result` panic

`argv.nth(1)  Option  arg.parse()  Result` `Option  Result` `Option  Result` `env::args()` `String`

```
use std::env;

fn double_arg(mut argv: env::Args) -> Result<i32, String> {
    argv.nth(1)
        .ok_or("Please give at least one argument".to_owned())
        .and_then(|arg| arg.parse::<i32>().map_err(|err| err.to_string()))
        .map(|n| 2 * n)
}

fn main() {
    match double_arg(env::args()) {
        Ok(n) => println!("{}", n),
        Err(err) => println!("Error: {}", err),
    }
}
```

`Option::ok_or` `Option  Result` `Option  None`

```
fn ok_or<T, E>(option: Option<T>, err: E) -> Result<T, E> {
    match option {
        Some(val) => Ok(val),
        None => Err(err),
    }
}
```

`Result::map_err` `Result::map  Result` error `Result  Ok(...)`

`map_err` `and_then` `Option<String>` `argv.nth(1)  Result<String, String>` `arg.parse()  ParseIntError  String`

IO Rust IO

`2`

`unwrap  unwrap`

```
use std::fs::File;
use std::io::Read;
use std::path::Path;

fn file_double<P: AsRef<Path>>(file_path: P) -> i32 {
    let mut file = File::open(file_path).unwrap(); // error 1
    let mut contents = String::new();
    file.read_to_string(&mut contents).unwrap(); // error 2
    let n: i32 = contents.trim().parse().unwrap(); // error 3
    2 * n
}

fn main() {
    let doubled = file_double("foobar");
    println!("{}", doubled);
}
```

`AsRef<Path>` `std::fs::File::open` bound

1.
2.
3.

`std::io::Error` `std::fs::File::open` `std::io::Read::read_to_string` `Result` `Result` `io::Error` `std::num::ParseIntError` `io::Error`

`file_double` panic `file_double` `i32` `i32`

`Option` `Result` `Option` `None` panic `Result<i32, E>` `E` `String`

```rust
use std::fs::File;
use std::io::Read;
use std::path::Path;

fn file_double<P: AsRef<Path>>(file_path: P) -> Result<i32, String> {
    File::open(file_path)
         .map_err(|err| err.to_string())
         .and_then(|mut file| {
              let mut contents = String::new();
              file.read_to_string(&mut contents)
                  .map_err(|err| err.to_string())
                  .map(|_| contents)
         })
         .and_then(|contents| {
              contents.trim().parse::<i32>()
                      .map_err(|err| err.to_string())
         })
         .map(|n| 2 * n)
}

fn main() {
    match file_double("foobar") {
        Ok(n) => println!("{}", n),
        Err(err) => println!("Error: {}", err),
    }
}
```

following the types `file_double` `Result<i32, String>` `and_then` `map` `map_err`

`and_then` `and_then`

`map` `Result` `Ok(...)` `map` `Ok(...)` `i32` `2` `map`

`map_err` `map_err` `map` `Result` `Err(...)` `String`
`io::Error` `num::ParseIntError` `ToString` `to_string()`

`file_double` case analysis

```rust
use std::fs::File;
use std::io::Read;
use std::path::Path;

fn file_double<P: AsRef<Path>>(file_path: P) -> Result<i32, String> {
    let mut file = match File::open(file_path) {
        Ok(file) => file,
        Err(err) => return Err(err.to_string()),
    };
    let mut contents = String::new();
    if let Err(err) = file.read_to_string(&mut contents) {
        return Err(err.to_string());
    }
    let n: i32 = match contents.trim().parse() {
        Ok(n) => n,
        Err(err) => return Err(err.to_string()),
    };
    Ok(2 * n)
}

fn main() {
    match file_double("foobar") {
        Ok(n) => println!("{}", n),
        Err(err) => println!("Error: {}", err),
    }
}
```

`match` `if let` case analysis

case analysis case analysis case analysis

`try!`

Rust `try!` `try!` case analysis

`try!`

```rust
macro_rules! try {
    ($e:expr) => (match $e {
        Ok(val) => val,
        Err(err) => return Err(err),
    });
}
```

`try!` case analysis

```rust
use std::fs::File;
use std::io::Read;
use std::path::Path;

fn file_double<P: AsRef<Path>>(file_path: P) -> Result<i32, String> {
    let mut file = try!(File::open(file_path).map_err(|e| e.to_string()));
    let mut contents = String::new();
    try!(file.read_to_string(&mut contents).map_err(|e| e.to_string()));
    let n = try!(contents.trim().parse::<i32>().map_err(|e| e.to_string()));
    Ok(2 * n)
}

fn main() {
    match file_double("foobar") {
        Ok(n) => println!("{}", n),
        Err(err) => println!("Error: {}", err),
    }
}
```

`try!` `map_err` `String` `map_err` `map_err` trait

trait `String`

`String` `String`

qiangpozheng

`String` `String`

`io::Error` `io::ErrorKind` IO `BrokenPipe` `NotFound` `io::ErrorKind` case analysis `String`

`String`

`enum` `io::Error` `num::ParseIntError`

```rust
use std::io;
use std::num;

// We derive `Debug` because all types should probably derive `Debug`.
// This gives us a reasonable human readable description of `CliError` values.
#[derive(Debug)]
enum CliError {
    Io(io::Error),
    Parse(num::ParseIntError),
}
```

`CliError`

```
# #[derive(Debug)]
# enum CliError { Io(::std::io::Error), Parse(::std::num::ParseIntError) }
use std::fs::File;
use std::io::Read;
use std::path::Path;

fn file_double<P: AsRef<Path>>(file_path: P) -> Result<i32, CliError> {
    let mut file = try!(File::open(file_path).map_err(CliError::Io));
    let mut contents = String::new();
    try!(file.read_to_string(&mut contents).map_err(CliError::Io));
    let n: i32 = try!(contents.trim().parse().map_err(CliError::Parse));
    Ok(2 * n)
}

fn main() {
    match file_double("foobar") {
        Ok(n) => println!("{}", n),
        Err(err) => println!("Error: {:?}", err),
    }
}
```

`map_err(|e| e.to_string())` `map_err(CliError::Io)` `map_err(CliError::Parse)` `String` `CliError` `enum`

`String`

trait

trait `https://github.com/rust-lang/rust/blob/master/src/doc/std/error/trait.Error.html` `std::convert::From` `Error` `From` trait

`Error` trait

`Error` trait

```
use std::fmt::{Debug, Display};

trait Error: Debug + Display {
  /// A short description of the error.
  fn description(&self) -> &str;

  /// The lower level cause of this error, if any.
  fn cause(&self) -> Option<&Error> { None }
}
```

trait trait

- `Debug`

- `Display`
- `description`
- `cause`

`Error` `Debug` `Display` `Error` `Error` `Error` trait `Box<Error>` `&Error` , `cause` `&Error` trait `Error` trait

`Error` trait

```
use std::io;
use std::num;

// We derive `Debug` because all types should probably derive `Debug`.
// This gives us a reasonable human readable description of `CliError` values.
#[derive(Debug)]
enum CliError {
    Io(io::Error),
    Parse(num::ParseIntError),
}
```

`From` trait

`try!` macro

`Box<Error>`

> choosing-your-guarantees.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust

Rust “wrapper ”

Rust

`Box<T>`

Box\“”“”

```
let x = Box::new(1);
let y = x;
// x no longer accessible here
```

`y` `x` `x`

`&T` `&mut T`

“”

`*const T` `*mut T`

C `unsafe`

`Vec<T>`

`Rc<T>`

Rc\“”

“”“refcount” `Rc` `Rc` `Rc<T>`

`Rc<T>`

`&T` `&T`

Rust

`Weak<T>` `&T` -- `Weak<T>` `None` `Rc`

`Rc<T>` `Box<T>` `usize` “”“”

`Rc<T>` / `Rc<T>`

Cell

`Cell` `&` `Rc<T>`

`cell`

`Cell<T>`

Cell\ `Copy`

```
use std::cell::Cell;

let x = Cell::new(1);
let y = &x;
let z = &x;
x.set(2);
y.set(3);
z.set(4);
println!("{}", x.get());
```

```
let mut x = 1;
let y = &mut x;
let z = &mut x;
x = 2;
*y = 3;
*z = 4;
println!("{}", x);
```

“” `Cell`

`Copy` `&` `&mut`

`Cell`

`Cell<T>` `Copy` `Cell<T>`

`RefCell<T>`

RefCell\ `Copy`

`RefCell<T>` `&T` / `&mut T` `borrow()` `borrow_mut()`

```
use std::cell::RefCell;

let x = RefCell::new(vec![1,2,3,4]);
{
    println!("{:?}", *x.borrow())
}

{
    let mut my_ref = x.borrow_mut();
    my_ref.push(1);
}
```

`Cell`

`RefCell` Rust `ctxt` map `&` `RefCell` cell

ServoDOMDOMDOM `RefCell` `Cell`

`&` `RefCell`

`RefCell`

`RefCell` “”

/

Synchronous types

`Rc<T>` `RefCell<T>` `Arc<T>` `Mutex<T>` / `RWLock<T>`

`Arc<T>`

Arc\ `Rc<T>` “Arc”

C++ `shared_ptr` `Arc` C++C++ `Arc<Mutex<T>>` `Arc<RwLock<T>>` `Arc<UnsafeCell<T>>`[4] `vec.push()` `UnsafeCell`

> 4. `Arc<UnsafeCell<T>>` `UnsafeCell<T>` `Send` `Sync` wrap `Send` / `Sync` `Arc<Wrapper<T>>` `Wrapper` `struct Wrapper<T>(UnsafeCell<T>)` ↩

`Rc` `Arc`

`Arc` `&`

`Mutex<T>` `RwLock<T>`

Mutex\RwLock\RAII guardguardmutex `lock()` guardguardguard

```
{
    let guard = mutex.lock();
    // guard dereferences mutably to the inner type
    *guard += 1;
} // lock released when destructor runs
```

`RwLock` writerreader `RwLock` reader“”writer“”reader

Composition

Rust `Rc<RefCell<Vec<T>>>`

`Rc<RefCell<T>>` `Rc<T>` `Rc<T>` `RefCell<T>` writerreaderreader

`Rc<RefCell<Vec<T>>>` `Rc<Vec<RefCell<T>>>` vector

`RefCell<T>` `Vec<T>` `Vec<T>` `Vec` `Rc` vector `Vec<T>` `&mut Vec<T>`

vectorvector `&mut [T]`[2]

`Arc<Mutex<T>>`

/

`Vec<RefCell<T>>` `RefCell<Vec<T>>`

1. `&[T]` `&mut [T]` slicevector `&mut [T]` ↩

(FFI)

> ffi.md
> commit 077f4eeb8485e5a1437f6e27973a907ac772b616

snappy/ Rust Rust C++ snappy C snappy-c.h

libc

`libc` crate C `Cargo.toml libc`

```
[dependencies]
libc = "0.2.0"
```

crate `extern crate libc;`

snappy

```
# #![feature(libc)]
extern crate libc;
use libc::size_t;

#[link(name = "snappy")]
extern {
    fn snappy_max_compressed_length(source_length: size_t) -> size_t;
}

fn main() {
    let x = unsafe { snappy_max_compressed_length(100) };
    println!("max compressed length of a 100 byte buffer: {}", x);
}
```

`extern` C ABI `#[link(...)]` snappy

`unsafe {}` CRust

Rust

`extern` snappy API

```
# #![feature(libc)]
extern crate libc;
use libc::{c_int, size_t};

#[link(name = "snappy")]
extern {
    fn snappy_compress(input: *const u8,
                       input_length: size_t,
                       compressed: *mut u8,
                       compressed_length: *mut size_t) -> c_int;
    fn snappy_uncompress(compressed: *const u8,
                         compressed_length: size_t,
                         uncompressed: *mut u8,
                         uncompressed_length: *mut size_t) -> c_int;
    fn snappy_max_compressed_length(source_length: size_t) -> size_t;
    fn snappy_uncompressed_length(compressed: *const u8,
                                  compressed_length: size_t,
                                  result: *mut size_t) -> c_int;
    fn snappy_validate_compressed_buffer(compressed: *const u8,
                                         compressed_length: size_t) -> c_int;
}
# fn main() {}
```

C API

`slice::raw` RustRust

```
# #![feature(libc)]
# extern crate libc;
# use libc::{c_int, size_t};
# unsafe fn snappy_validate_compressed_buffer(_: *const u8, _: size_t) -> c_int { 0 }
# fn main() {}
pub fn validate_compressed_buffer(src: &[u8]) -> bool {
    unsafe {
        snappy_validate_compressed_buffer(src.as_ptr(), src.len() as size_t) == 0
    }
}
```

`validate_compressed_buffer` `unsafe` `unsafe`

`snappy_compress` `snappy_uncompress`

`snappy_max_compressed_length` `snappy_compress`

```
# #![feature(libc)]
# extern crate libc;
# use libc::{size_t, c_int};
# unsafe fn snappy_compress(a: *const u8, b: size_t, c: *mut u8,
#                           d: *mut size_t) -> c_int { 0 }
# unsafe fn snappy_max_compressed_length(a: size_t) -> size_t { a }
# fn main() {}
pub fn compress(src: &[u8]) -> Vec<u8> {
    unsafe {
        let srclen = src.len() as size_t;
        let psrc = src.as_ptr();

        let mut dstlen = snappy_max_compressed_length(srclen);
        let mut dst = Vec::with_capacity(dstlen as usize);
        let pdst = dst.as_mut_ptr();

        snappy_compress(psrc, srclen, pdst, &mut dstlen);
        dst.set_len(dstlen as usize);
        dst
    }
}
```

snappy `snappy_uncompressed_length`

```rust
# #![feature(libc)]
# extern crate libc;
# use libc::{size_t, c_int};
# unsafe fn snappy_uncompress(compressed: *const u8,
#                              compressed_length: size_t,
#                              uncompressed: *mut u8,
#                              uncompressed_length: *mut size_t) -> c_int { 0 }
# unsafe fn snappy_uncompressed_length(compressed: *const u8,
#                                       compressed_length: size_t,
#                                       result: *mut size_t) -> c_int { 0 }
# fn main() {}
pub fn uncompress(src: &[u8]) -> Option<Vec<u8>> {
    unsafe {
        let srclen = src.len() as size_t;
        let psrc = src.as_ptr();

        let mut dstlen: size_t = 0;
        snappy_uncompressed_length(psrc, srclen, &mut dstlen);

        let mut dst = Vec::with_capacity(dstlen as usize);
        let pdst = dst.as_mut_ptr();

        if snappy_uncompress(psrc, srclen, pdst, &mut dstlen) == 0 {
            dst.set_len(dstlen as usize);
            Some(dst)
        } else {
            None // SNAPPY_INVALID_INPUT
        }
    }
}
```

GitHub

Rust

`Drop` trait

RustCCallbacks from C code to Rust functions

Rust `extern` C

C

Rust

```rust
extern fn callback(a: i32) {
    println!("I'm called from C with value {0}", a);
}

#[link(name = "extlib")]
extern {
   fn register_callback(cb: extern fn(i32)) -> i32;
   fn trigger_callback();
}

fn main() {
    unsafe {
        register_callback(callback);
        trigger_callback(); // Triggers the callback
    }
}
```

C

```c
typedef void (*rust_callback)(int32_t);
rust_callback cb;

int32_t register_callback(rust_callback callback) {
    cb = callback;
    return 1;
}

void trigger_callback() {
  cb(7); // Will call callback(7) in Rust
}
```

Rust `main()` C `trigger_callback()` Rust `callback()`

RustTargeting callbacks to Rust objects

CRustC

CCRustRust

Rust

```
#[repr(C)]
struct RustObject {
    a: i32,
    // other members
}

extern "C" fn callback(target: *mut RustObject, a: i32) {
    println!("I'm called from C with value {0}", a);
    unsafe {
        // Update the value in RustObject with the value received from the callback
        (*target).a = a;
    }
}

#[link(name = "extlib")]
extern {
   fn register_callback(target: *mut RustObject,
                        cb: extern fn(*mut RustObject, i32)) -> i32;
   fn trigger_callback();
}

fn main() {
    // Create the object that will be referenced in the callback
    let mut rust_object = Box::new(RustObject { a: 5 });

    unsafe {
        register_callback(&mut *rust_object, callback);
        trigger_callback();
    }
}
```

C

```
typedef void (*rust_callback)(void*, int32_t);
void* cb_target;
rust_callback cb;

int32_t register_callback(void* callback_target, rust_callback callback) {
    cb_target = callback_target;
    cb = callback;
    return 1;
}

void trigger_callback() {
  cb(cb_target, 7); // Will call callback(&rustObject, 7) in Rust
}
```

C Rust C Rust

Rust `std::comm` C Rust

Rust Rust Rust C

`extern link rustc`

- `#[link(name = "foo")]`
- `#[link(name = "foo", kind = "bar")]`

`foo` `bar` 3

- - `#[link(name = "readline")]`
- - `#[link(name = "my_build_dependency", kind = "static")]`
- - `#[link(name = "CoreFoundation", kind = "framework")]`

OSX

`kind` rustrlib/staticlibdylib/binary

- RustC/C++C/C++ `libfoo.a` rust `#[link(name = "foo", kind = "static")]`

- `readline` rlibsrlib

OSX

Unsafe blocks

unsafeunsafe

```
unsafe fn kaboom(ptr: *const int) -> int { *ptr }
```

`unsafe unsafe`

Accessing foreign globals

API `extern static`

```
# #![feature(libc)]
extern crate libc;

#[link(name = "readline")]
extern {
    static rl_readline_version: libc::c_int;
}

fn main() {
    println!("You have readline version {} installed.",
             rl_readline_version as i32);
}
```

`mut`

```
# #![feature(libc)]
extern crate libc;

use std::ffi::CString;
use std::ptr;

#[link(name = "readline")]
extern {
    static mut rl_prompt: *const libc::c_char;
}

fn main() {
    let prompt = CString::new("[my-awesome-shell] $").unwrap();
    unsafe {
        rl_prompt = prompt.as_ptr();

        println!("{:?}", rl_prompt);

        rl_prompt = ptr::null();
    }
}
```

`static mut`

Foreign calling conventions

CABIRustCWindows APIRust

```
# #![feature(libc)]
extern crate libc;

#[cfg(all(target_os = "win32", target_arch = "x86"))]
#[link(name = "kernel32")]
#[allow(non_snake_case)]
extern "stdcall" {
    fn SetEnvironmentVariableA(n: *const u8, v: *const u8) -> libc::c_int;
}
# fn main() { }
```

`extern` ABI

- `stdcall`
- `aapcs`
- `cdecl`
- `fastcall`
- `vectorcall` `abi_vectorcall` gate
- `Rust`
- `rust-intrinsic`
- `system`
- `C`
- `win64`

ABI `system` ABIABIx86win32 `stdcall` ABIx86_64windows `C` `C` `extern "system" { ... }` windows x86

Interoperability with foreign code

`#[repr(C)]` Rust `struct` C `#[repr(C, packed)]` `#[repr(C)]`

Rust `Box<T>` `*`

`vec` `str` C API `\0` NUL C `std::ffi` `CString`

`libc` CRust `libc` `libm`

“”The "nullable pointer optimization"

`nulll` `&T` `&mut T` `Box<T>` `extern "abi" fn()` C `enum` “” `Option<extern "C" fn(c_int) -> c_int>` C ABI

C Rust

Rus t C

```
#[no_mangle]
pub extern fn hello_rust() -> *const u8 {
    "Hello, world!\0".as_ptr()
}
# fn main() {}
```

`extern` C `no_mangle` Rust

FFI panic

FFI `panic!` FFI `panic!` panicpanicC

```
use std::thread;

#[no_mangle]
pub extern fn oh_no() -> i32 {
    let h = thread::spawn(|| {
        panic!("Oops!");
    });

    match h.join() {
        Ok(_) => 1,
        Err(_) => 0,
    }
}
# fn main() {}
```

opaque

C `void *`

```
void foo(void *arg);
void bar(void *arg);
```

`c_void` Rust

```
# #![feature(libc)]
extern crate libc;

extern "C" {
    pub fn foo(arg: *mut libc::c_void);
    pub fn bar(arg: *mut libc::c_void);
}
# fn main() {}
```

C `struct` `opaque` C

```c
struct Foo; /* Foo is a structure, but its contents are not part of the public interfac
e */
struct Bar;
void foo(struct Foo *arg);
void bar(struct Bar *arg);
```

Rust `enum` opaque

```rust
pub enum Foo {}
pub enum Bar {}

extern "C" {
    pub fn foo(arg: *mut Foo);
    pub fn bar(arg: *mut Bar);
}
# fn main() {}
```

`enum` opaque `Foo` `Bar` `bar()` `Foo`

`Borrow` `AsRef`

> borrow-and-asref.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

`Borrow` `AsRef`

`Borrow`

`Borrow`

`HashMap` `Borrow` `get`

```rust
fn get<Q: ?Sized>(&self, k: &Q) -> Option<&V>
    where K: Borrow<Q>,
          Q: Hash + Eq
```

`k` `HashMap`

```rust
struct HashMap<K, V, S = RandomState> {
```

`k` `HashMap` `key` `get()` `Borrow<Q>` `get()` `HashMap` `String` `&str`

```rust
use std::collections::HashMap;

let mut map = HashMap::new();
map.insert("Foo".to_string(), 42);

assert_eq!(map.get("Foo"), Some(&42));
```

`String` `Borrow<str>`

`&T` `Borrow` `&[T]` `&mut [T]` `Borrow`

```rust
use std::borrow::Borrow;
use std::fmt::Display;

fn foo<T: Borrow<i32> + Display>(a: T) {
    println!("a is borrowed: {}", a);
}

let mut i = 5;

foo(&i);
foo(&mut i);
```

`a is borrowed: 5`

`AsRef`

`AsRef`

```rust
let s = "Hello".to_string();

fn foo<T: AsRef<str>>(s: T) {
    let slice = s.as_ref();
}
```

`Borrow`

`AsRef`

> release-channels.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust “”Rust

Rust 3

- Nightly
- Beta
- Stable

6“”6“” `1.x`

6 `1.x` `1.(x + 1)-beta` `1.(x + 2)-nightly`

RustRustRust

CI

Rustregression

Rust Traviscrate Travis Rust `.travis.yml`

```
language: rust
rust:
  - nightly
  - beta
  - stable

matrix:
  allow_failures:
    - rust: nightly
```

Travis CI CI

Rust

> nightly-rust.md
>
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust3betaRust

Rust `rustup.sh`

```
$ curl -s https://static.rust-lang.org/rustup.sh | sudo sh -s -- --channel=nightly
```

`curl | sudo sh`

```
$ curl -L https://static.rust-lang.org/rustup.sh -O
$ sudo sh rustup.sh
```

Windows 3264

Rust:(Rust

```
$ sudo /usr/local/lib/rustlib/uninstall.sh
```

Windows .exe

RustRustRust1.0Rust

`curl | sudo sh` Rust Rust

- Windows (7, 8, Server 2008 R2)
- Linux (2.6.18 or later, various distributions), x86 and x86-64
- OSX 10.7 (Lion) or greater, x86 and x86-64

Rust Android

WindowsRustWindowsWIndowsLinux/OS XbugWindows

RustShell

```
$ rustc --version
```

hash

```
rustc 1.0.0-nightly (f11f3e7ba 2015-01-04) (built 2015-01-06)
```

Rust

MibbitRust IRC irc.mozilla.org RustaceansRuster the /r/rust subredditStack Overflow

> compiler-plugins.md
> commit 1430a3500076ad504a0b30be77fd2ad4468ea769

`rustc` lint

`rustc` *registrar* `#![plugin(...)]` rustc::plugin

`#![plugin(foo(... args ...))]` `rustc` `Registry` args

`#![plugin]` `extern crate` `libsyntax` `librustc` `plugin_as_library` lint

`macro_rules!` Rust

Rust Rust

roman_numerals.rs

```rust
#![crate_type="dylib"]
#![feature(plugin_registrar, rustc_private)]

extern crate syntax;
extern crate rustc;

use syntax::codemap::Span;
use syntax::parse::token;
use syntax::ast::{TokenTree, TtToken};
use syntax::ext::base::{ExtCtxt, MacResult, DummyResult, MacEager};
use syntax::ext::build::AstBuilder;  // trait for expr_usize
use rustc::plugin::Registry;

fn expand_rn(cx: &mut ExtCtxt, sp: Span, args: &[TokenTree])
        -> Box<MacResult + 'static> {

    static NUMERALS: &'static [(&'static str, u32)] = &[
        ("M", 1000), ("CM", 900), ("D", 500), ("CD", 400),
        ("C",  100), ("XC",  90), ("L",  50), ("XL",  40),
        ("X",   10), ("IX",   9), ("V",   5), ("IV",   4),
        ("I",    1)];

    let text = match args {
        [TtToken(_, token::Ident(s, _))] => token::get_ident(s).to_string(),
        _ => {
            cx.span_err(sp, "argument should be a single identifier");
            return DummyResult::any(sp);
        }
    };

    let mut text = &*text;
    let mut total = 0;
    while !text.is_empty() {
        match NUMERALS.iter().find(|&&(rn, _)| text.starts_with(rn)) {
            Some(&(rn, val)) => {
                total += val;
                text = &text[rn.len()..];
            }
            None => {
                cx.span_err(sp, "invalid Roman numeral");
                return DummyResult::any(sp);
            }
        }
    }

    MacEager::expr(cx.expr_u32(sp, total))
}

#[plugin_registrar]
pub fn plugin_registrar(reg: &mut Registry) {
    reg.register_macro("rn", expand_rn);
}
```

`rn!()`

```
#![feature(plugin)]
#![plugin(roman_numerals)]

fn main() {
    assert_eq!(rn!(MMXV), 2015);
}
```

`fn(&str) -> u32`

-
-
-

derive Registry::register_syntax_extensionSyntaxExtension enum regex_macros

syntax::parse

```
fn expand_foo(cx: &mut ExtCtxt, sp: Span, args: &[TokenTree])
        -> Box<MacResult+'static> {

    let mut parser = cx.new_parser_from_tts(args);

    let expr: P<Expr> = parser.parse_expr();
```

libsyntax

Span   Spanned

ExtCtxt::span_fatal   ExtCtxt::span_errDummyResult

span_notesyntax::print::pprust::*_to_string

AstBuilder::expr_usize `AstBuilder` `libsyntax`

Lint

Rust Lint   src/test/auxiliary/lint_plugin_test.rs

```
declare_lint!(TEST_LINT, Warn,
              "Warn about items named 'lintme'")

struct Pass;

impl LintPass for Pass {
    fn get_lints(&self) -> LintArray {
        lint_array!(TEST_LINT)
    }

    fn check_item(&mut self, cx: &Context, it: &ast::Item) {
        let name = token::get_ident(it.ident);
        if name.get() == "lintme" {
            cx.span_lint(TEST_LINT, it.span, "item is named 'lintme'");
        }
    }
}

#[plugin_registrar]
pub fn plugin_registrar(reg: &mut Registry) {
    reg.register_lint_pass(box Pass as LintPassObject);
}
```

```
#![plugin(lint_plugin_test)]

fn lintme() { }
```

```
foo.rs:4:1: 4:16 warning: item is named 'lintme', #[warn(test_lint)] on by default
foo.rs:4 fn lintme() { }
         ^~~~~~~~~~~~~~~
```

Lint

- `declare_lint!` Lint
- lint
- LintPass `LintPass` `Lint` `span_lint` `get_lints`

Lint `rustc` lintlint

`[#[allow(test_lint)]]` `-A test-lint` `declare_lint!`

`rustc -W help foo.rs` lint `rustc` `foo.rs`

> inline-assembly.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

CPURust `asm!` GCC Clang

```
asm!(assembly template
   : output operands
   : input operands
   : clobbers
   : options
   );
```

`asm` `#![feature(asm)]` `unsafe`

> x86/x86-64

`assembly template` `""`

```
#![feature(asm)]

#[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
fn foo() {
    unsafe {
        asm!("NOP");
    }
}

// other platforms
#[cfg(not(any(target_arch = "x86", target_arch = "x86_64")))]
fn foo() { /* ... */ }

fn main() {
    // ...
    foo();
    // ...
}
```

`feature(asm)` `#[cfg]`

`:`

```
# #![feature(asm)]
# #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
# fn main() { unsafe {
asm!("xor %eax, %eax"
    :
    :
    : "{eax}"
   );
# } }
```

```
# #![feature(asm)]
# #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
# fn main() { unsafe {
asm!("xor %eax, %eax" ::: "{eax}");
# } }
```

`: "constraints1"(expr1), "constraints2"(expr2), ..."`

```
# #![feature(asm)]
# #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
fn add(a: i32, b: i32) -> i32 {
    let c: i32;
    unsafe {
        asm!("add $2, $0"
             : "=r"(c)
             : "0"(a), "r"(b)
             );
    }
    c
}
# #[cfg(not(any(target_arch = "x86", target_arch = "x86_64")))]
# fn add(a: i32, b: i32) -> i32 { a + b }

fn main() {
    assert_eq!(add(3, 14159), 14162)
}
```

`{}`

```
# #![feature(asm)]
# #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
# unsafe fn read_byte_in(port: u16) -> u8 {
let result: u8;
asm!("in %dx, %al" : "={al}"(result) : "{dx}"(port));
result
# }
```

Clobbers

```
# #![feature(asm)]
# #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
# fn main() { unsafe {
// Put the value 0x200 in eax
asm!("mov $$0x200, %eax" : /* no outputs */ : /* no inputs */ : "{eax}");
# } }
```

`cc` `memory`

Options

`options` Rust `:"foo", "bar", "baz"`

1. *volatile* - gcc/clang `__asm__ __volatile__ (...)`
2. *alignstack* - SSE
3. *intel* - intel AT&T

```
# #![feature(asm)]
# #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
# fn main() {
let result: i32;
unsafe {
   asm!("mov eax, 2" : "={eax}"(result) : : : "intel")
}
println!("eax is currently {}", result);
# }
```

`asm!` LLVM

> no-stdlib.md
> commit 0394418752cd39c5da68e7e05d5a37bf5a30f0db

Rust host Rust Rust `#![no_std]`

> `#![no_std]` `#![no_std]`

`#[start]` C

```
# #![feature(libc)]
#![feature(lang_items)]
#![feature(start)]
#![no_std]

// Pull in the system libc library for what crt0.o likely requires
extern crate libc;

// Entry point for this program
#[start]
fn start(_argc: isize, _argv: *const *const u8) -> isize {
    0
}

// These functions and traits are used by the compiler, but not
// for a bare-bones hello world. These are normally
// provided by libstd.
#[lang = "eh_personality"] extern fn eh_personality() {}
#[lang = "panic_fmt"] fn panic_fmt() -> ! { loop {} }
# #[lang = "eh_unwind_resume"] extern fn rust_eh_unwind_resume() {}
# #[no_mangle] pub extern fn rust_eh_register_frames () {}
# #[no_mangle] pub extern fn rust_eh_unregister_frames () {}
# // fn main() {} tricked you, rustdoc!
```

`main` shim `#![no_main]` ABI

```
# #![feature(libc)]
#![feature(lang_items)]
#![feature(start)]
#![no_std]
#![no_main]

extern crate libc;

#[no_mangle] // ensure that this symbol is called `main` in the output
pub extern fn main(argc: i32, argv: *const *const u8) -> i32 {
    0
}

#[lang = "eh_personality"] extern fn eh_personality() {}
#[lang = "panic_fmt"] fn panic_fmt() -> ! { loop {} }
# #[lang = "eh_unwind_resume"] extern fn rust_eh_unwind_resume() {}
# #[no_mangle] pub extern fn rust_eh_register_frames () {}
# #[no_mangle] pub extern fn rust_eh_unregister_frames () {}
# // fn main() {} tricked you, rustdoc!
```

`stack_exhausted` ”“

`eh_personality` GCC libstd `panic_fmt`

libcore

RustRust libcore

Rust

CRust

```rust
#![feature(lang_items, start, no_std, core, libc)]
#![no_std]

extern crate core;

use core::prelude::*;

use core::mem;

#[no_mangle]
pub extern fn dot_product(a: *const u32, a_len: u32,
                          b: *const u32, b_len: u32) -> u32 {
    use core::raw::Slice;

    // Convert the provided arrays into Rust slices.
    // The core::raw module guarantees that the Slice
    // structure has the same memory layout as a &[T]
    // slice.
    //
    // This is an unsafe operation because the compiler
    // cannot tell the pointers are valid.
    let (a_slice, b_slice): (&[u32], &[u32]) = unsafe {
        mem::transmute((
            Slice { data: a, len: a_len as usize },
            Slice { data: b, len: b_len as usize },
        ))
    };

    // Iterate over the slices, collecting the result
    let mut ret = 0;
    for (i, j) in a_slice.iter().zip(b_slice.iter()) {
        ret += (*i) * (*j);
    }
    return ret;
}

#[lang = "panic_fmt"]
extern fn panic_fmt(args: &core::fmt::Arguments,
                    file: &str,
                    line: u32) -> ! {
    loop {}
}

#[lang = "stack_exhausted"] extern fn stack_exhausted() {}
#[lang = "eh_personality"] extern fn eh_personality() {}
```

`lang` `panic_fmt` libcore `panic_fmt`

Rust `liballoc` libcore

> intrinsics.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d
>
> libcore

FFI `rust-intrinsic` ABI `transmute`

```
#![feature(intrinsics)]
# fn main() {}

extern "rust-intrinsic" {
    fn transmute<T, U>(x: T) -> U;

    fn offset<T>(dst: *const T, offset: isize) -> *const T;
}
```

FFI `unsafe`

> lang-items.md
> commit 464cdff102993ff1900eebbf65209e0a3c0be0d5
>
> Rust crate crate

`rustc` `#[lang="..."]` `...` ,“”

`Box` `Box` `malloc` `free`

```
#![feature(lang_items, box_syntax, start, libc)]
#![no_std]

extern crate libc;

extern {
    fn abort() -> !;
}

#[lang = "owned_box"]
pub struct Box<T>(*mut T);

#[lang = "exchange_malloc"]
unsafe fn allocate(size: usize, _align: usize) -> *mut u8 {
    let p = libc::malloc(size as libc::size_t) as *mut u8;

    // malloc failed
    if p as usize == 0 {
        abort();
    }

    p
}
#[lang = "exchange_free"]
unsafe fn deallocate(ptr: *mut u8, _size: usize, _align: usize) {
    libc::free(ptr as *mut libc::c_void)
}

#[start]
fn main(argc: isize, argv: *const *const u8) -> isize {
    let x = box 1;

    0
}

#[lang = "eh_personality"] extern fn eh_personality() {}
#[lang = "panic_fmt"] fn panic_fmt() -> ! { loop {} }
# #[lang = "eh_unwind_resume"] extern fn rust_eh_unwind_resume() {}
# #[no_mangle] pub extern fn rust_eh_register_frames () {}
# #[no_mangle] pub extern fn rust_eh_unregister_frames () {}
```

`abort` `exchange_malloc`

- `==` `<` `*` `+` 4 `eq` `ord` `deref` `add`
- `eh_personality` `fail` `fail_bounds_checks`
- `std::marker` `send` `sync` `copy`
- `std::marker` `covariant_type` `contravariant_lifetime`

`Box` `exchange_malloc` `exchange_free` `rustc`

> advanced-linking.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust Rust

rustc `link_args` `extern`

```
#![feature(link_args)]

#[link_args = "-foo -bar -baz"]
extern {}
# fn main() {}
```

`feature(link_args)` gate rustc shell `gcc` MSVC `link.exe` rustc LLVM `link_args` `rustc -C link-args link_args`

`[link(...)] extern`

Rust Rust `libc libm`

--

Linux

Linux Rust `libc` GCC `glibc` 64 Linux Linux `libc`

```
$ mkdir musldist
$ PREFIX=$(pwd)/musldist
$
$ # Build musl
$ curl -O http://www.musl-libc.org/releases/musl-1.1.10.tar.gz
$ tar xf musl-1.1.10.tar.gz
$ cd musl-1.1.10/
musl-1.1.10 $ ./configure --disable-shared --prefix=$PREFIX
musl-1.1.10 $ make
musl-1.1.10 $ make install
musl-1.1.10 $ cd ..
$ du -h musldist/lib/libc.a
2.2M    musldist/lib/libc.a
$
$ # Build libunwind.a
$ curl -O http://llvm.org/releases/3.7.0/llvm-3.7.0.src.tar.xz
$ tar xf llvm-3.7.0.src.tar.xz
$ cd llvm-3.7.0.src/projects/
llvm-3.7.0.src/projects $ curl http://llvm.org/releases/3.7.0/libunwind-3.7.0.src.tar.x
z | tar xJf -
llvm-3.7.0.src/projects $ mv libunwind-3.7.0.src libunwind
llvm-3.7.0.src/projects $ mkdir libunwind/build
llvm-3.7.0.src/projects $ cd libunwind/build
llvm-3.7.0.src/projects/libunwind/build $ cmake -DLLVM_PATH=../../.. -DLIBUNWIND_ENABLE
_SHARED=0 ..
llvm-3.7.0.src/projects/libunwind/build $ make
llvm-3.7.0.src/projects/libunwind/build $ cp lib/libunwind.a $PREFIX/lib/
llvm-3.7.0.src/projects/libunwind/build $ cd ../../../../
$ du -h musldist/lib/libunwind.a
164K    musldist/lib/libunwind.a
$
$ # Build musl-enabled rust
$ git clone https://github.com/rust-lang/rust.git muslrust
$ cd muslrust
muslrust $ ./configure --target=x86_64-unknown-linux-musl --musl-root=$PREFIX --prefix=
$PREFIX
muslrust $ make
muslrust $ make install
muslrust $ cd ..
$ du -h musldist/bin/rustc
12K     musldist/bin/rustc
```

`musl` Rust

```
$ export PATH=$PREFIX/bin:$PATH
$ export LD_LIBRARY_PATH=$PREFIX/lib:$LD_LIBRARY_PATH
$ echo 'fn main() { println!("hi!"); panic!("failed"); }' > example.rs
$ rustc --target=x86_64-unknown-linux-musl example.rs
$ ldd example
        not a dynamic executable
$ ./example
hi!
thread '<main>' panicked at 'failed', example.rs:1
```

Linux

`cargo build --target` crate `musl`

> benchmark-tests.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rust `src/lib.rs`

```
#![feature(test)]

extern crate test;

pub fn add_two(a: i32) -> i32 {
    a + 2
}

#[cfg(test)]
mod tests {
    use super::*;
    use test::Bencher;

    #[test]
    fn it_works() {
        assert_eq!(4, add_two(2));
    }

    #[bench]
    fn bench_add_two(b: &mut Bencher) {
        b.iter(|| add_two(2));
    }
}
```

`test` gate

`test` crate `bench` `&mut Bencher` `Bencher` `iter`

`cargo bench`

```
$ cargo bench
   Compiling adder v0.0.1 (file:///home/steve/tmp/adder)
     Running target/release/adder-91b3e234d4ed382a

running 2 tests
test tests::it_works ... ignored
test tests::bench_add_two ... bench:         1 ns/iter (+/- 0)

test result: ok. 0 passed; 0 failed; 1 ignored; 1 measured
```

`cargo bench` `cargo test` Rust `1 ns/iter (+/- 0)`

- `iter`
- “”
- idempotentrunner
- `iter`
- `iter`

Gocha

```
#![feature(test)]

extern crate test;
use test::Bencher;

#[bench]
fn bench_xor_1000_ints(b: &mut Bencher) {
    b.iter(|| {
        (0..1000).fold(0, |old, new| old ^ new);
    });
}
```

```
running 1 test
test bench_xor_1000_ints ... bench:         0 ns/iter (+/- 0)

test result: ok. 0 passed; 0 failed; 0 ignored; 1 measured
```

`iter` `b.iter`

```
# struct X;
# impl X { fn iter<T, F>(&self, _: F) where F: FnMut() -> T {} } let b = X;
b.iter(|| {
    // note lack of `;` (could also use an explicit `return`).
    (0..1000).fold(0, |old, new| old ^ new)
});
```

`test::black_box` “”

```
#![feature(test)]

extern crate test;

# fn main() {
# struct X;
# impl X { fn iter<T, F>(&self, _: F) where F: FnMut() -> T {} } let b = X;
b.iter(|| {
    let n = test::black_box(1000);

    (0..n).fold(0, |a, b| a ^ b)
})
# }
```

`black_box(&huge_struct)`

```
running 1 test
test bench_xor_1000_ints ... bench:       131 ns/iter (+/- 3)

test result: ok. 0 passed; 0 failed; 0 ignored; 1 measured
```

> box-syntax-and-patterns.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

`Box` `Box::new` `Box` `box` `Box`

```
#![feature(box_syntax, box_patterns)]

fn main() {
    let b = Some(box 5);
    match b {
        Some(box n) if n < 0 => {
            println!("Box contains negative number {}", n);
        },
        Some(box n) if n >= 0 => {
            println!("Box contains non-negative number {}", n);
        },
        None => {
            println!("No box");
        },
        _ => unreachable!()
    }
}
```

`box_syntax` `box_patterns` gate

```rust
struct BigStruct {
    one: i32,
    two: i32,
    // etc
    one_hundred: i32,
}

fn foo(x: Box<BigStruct>) -> Box<BigStruct> {
    Box::new(*x)
}

fn main() {
    let x = Box::new(BigStruct {
        one: 1,
        two: 2,
        one_hundred: 100,
    });

    let y = foo(x);
}
```

`BigStruct` `int`

Rust

```rust
#![feature(box_syntax)]

struct BigStruct {
    one: i32,
    two: i32,
    // etc
    one_hundred: i32,
}

fn foo(x: Box<BigStruct>) -> BigStruct {
    *x
}

fn main() {
    let x = Box::new(BigStruct {
        one: 1,
        two: 2,
        one_hundred: 100,
    });

    let y: Box<BigStruct> = box foo(x);
}
```

Rust `main` `foo` `x` `foo` `Box<T>`

slice-patterns.md
commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

`slice_patterns` `&`

```
#![feature(slice_patterns)]

fn main() {
    let v = vec!["match_this", "1"];

    match &v[..] {
        ["match_this", second] => println!("The second element is {}", second),
        _ => {},
    }
}
```

`advanced_slice_patterns` gate `..` `..`

```
#![feature(advanced_slice_patterns, slice_patterns)]

fn is_symmetric(list: &[u32]) -> bool {
    match list {
        [] | [_] => true,
        [x, inside.., y] if x == y => is_symmetric(inside),
        _ => false
    }
}

fn main() {
    let sym = &[0, 1, 4, 2, 4, 1, 0];
    assert!(is_symmetric(sym));

    let not_sym = &[0, 1, 7, 2, 4, 1, 0];
    assert!(!is_symmetric(not_sym));
}
```

> associated-constants.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

`associated_consts`

```
#![feature(associated_consts)]

trait Foo {
    const ID: i32;
}

impl Foo for i32 {
    const ID: i32 = 1;
}

fn main() {
    assert_eq!(1, i32::ID);
}
```

`Foo` `ID`

```
#![feature(associated_consts)]

trait Foo {
    const ID: i32;
}

impl Foo for i32 {
}
```

```
error: not all trait items implemented, missing: `ID` [E0046]
     impl Foo for i32 {
     }
```

```rust
#![feature(associated_consts)]

trait Foo {
    const ID: i32 = 1;
}

impl Foo for i32 {
}

impl Foo for i64 {
    const ID: i32 = 5;
}

fn main() {
    assert_eq!(1, i32::ID);
    assert_eq!(5, i64::ID);
}
```

`Foo` `i32` `i64`

trait `struct` `impl` `enum`

```rust
#![feature(associated_consts)]

struct Foo;

impl Foo {
    pub const FOO: u32 = 3;
}
```

> custom-allocators.md
> commit 6ba952020fbc91bad64be1ea0650bfba52e6aab4

Rust RFC 1183

`alloc_system` `alloc_jemalloc` jemalloc Rust crate routine

`alloc_jemalloc` “”

`alloc_system` Rust “” API `malloc` `free`

```rust
#![feature(alloc_system)]

extern crate alloc_system;

fn main() {
    let a = Box::new(4); // allocates from the system allocator
    println!("{}", a);
}
```

jemalloc jemalloc

```rust
#![feature(alloc_jemalloc)]
#![crate_type = "dylib"]

extern crate alloc_jemalloc;

pub fn foo() {
    let a = Box::new(4); // allocates from jemalloc
    println!("{}", a);
}
# fn main() {}
```

jemalloc API `alloc_system` `alloc_jemallo` crate `alloc_system`

```rust
# // only needed for rustdoc --test down below
# #![feature(lang_items)]
```

```
// The compiler needs to be instructed that this crate is an allocator in order
// to realize that when this is linked in another allocator like jemalloc should
// not be linked in
#![feature(allocator)]
#![allocator]

// Allocators are not allowed to depend on the standard library which in turn
// requires an allocator in order to avoid circular dependencies. This crate,
// however, can use all of libcore.
#![no_std]

// Let's give a unique name to our custom allocator
#![crate_name = "my_allocator"]
#![crate_type = "rlib"]

// Our system allocator will use the in-tree libc crate for FFI bindings. Note
// that currently the external (crates.io) libc cannot be used because it links
// to the standard library (e.g. `#![no_std]` isn't stable yet), so that's why
// this specifically requires the in-tree version.
#![feature(libc)]
extern crate libc;

// Listed below are the five allocation functions currently required by custom
// allocators. Their signatures and symbol names are not currently typechecked
// by the compiler, but this is a future extension and are required to match
// what is found below.
//
// Note that the standard `malloc` and `realloc` functions do not provide a way
// to communicate alignment so this implementation would need to be improved
// with respect to alignment in that aspect.

#[no_mangle]
pub extern fn __rust_allocate(size: usize, _align: usize) -> *mut u8 {
    unsafe { libc::malloc(size as libc::size_t) as *mut u8 }
}

#[no_mangle]
pub extern fn __rust_deallocate(ptr: *mut u8, _old_size: usize, _align: usize) {
    unsafe { libc::free(ptr as *mut libc::c_void) }
}

#[no_mangle]
pub extern fn __rust_reallocate(ptr: *mut u8, _old_size: usize, size: usize,
                                _align: usize) -> *mut u8 {
    unsafe {
        libc::realloc(ptr as *mut libc::c_void, size as libc::size_t) as *mut u8
    }
}

#[no_mangle]
pub extern fn __rust_reallocate_inplace(_ptr: *mut u8, old_size: usize,
                                        _size: usize, _align: usize) -> usize {
    old_size // this api is not supported by libc
}

#[no_mangle]
pub extern fn __rust_usable_size(size: usize, _align: usize) -> usize {
    size
}

# // only needed to get rustdoc to test this
# fn main() {}
# #[lang = "panic_fmt"] fn panic_fmt() {}
# #[lang = "eh_personality"] fn eh_personality() {}
# #[lang = "eh_unwind_resume"] extern fn eh_unwind_resume() {}
# #[no_mangle] pub extern fn rust_eh_register_frames () {}
# #[no_mangle] pub extern fn rust_eh_unregister_frames () {}
```

crate

```
extern crate my_allocator;

fn main() {
    let a = Box::new(8); // allocates memory via our custom allocator crate
    println!("{}", a);
}
```

- rlib
- `#![needs_allocator]` `liballoc` `#[allocator]` crate crate libcore

> glossary.md
> commit 024aa9a345e92aa1926517c4d9b16bd83e74c10d

Rustacean

Arity

Arity

```
let x = (2, 3);
let y = (4, 6);
let z = (8, 2, 6);
```

`x` `y` Arity `2` `z` Arity `3`

Abstract Syntax Tree

‘’‘AST’ `2 + 3`

```
   +
  / \
 2   3
```

`2 + (3 * 4)`

```
   +
  / \
 2   *
    / \
   3   4
```

Arity

Arity

```
let x = (2, 3);
let y = (4, 6);
let z = (8, 2, 6);
```

`x` `y` arity 2 `z` arity 3

Bounds

trait

DST (Dynamically Sized Type)

Expression

`2 + (3 * 4)` 14

Expression-Oriented Language

Statement

> syntax-index.md
> commit 1b438314a07d4cc2ecf0d82cd195e28bef73eac2

Keywords

- `as` : `as`
- `break` :
- `const` : `const` `static`
- `continue` :
- `crate` : crate crate crate
- `else` : `if` `if let` `if` `if let`
- `enum` :
- `extern` : crate crate crate
- `false` : false
- `fn` :
- `for` : `impl` trait for
- `if` : `if` `if let`
- `impl` : trait
- `in` : `for` for
- `let` :
- `loop` : loop
- `match` :
- `mod` : crate
- `move` : `move`
- `mut` :
- `pub` : `struct` `impl` crate
- `ref` : `ref` `ref mut`
- `return` :
- `Self` : trait Traits
- `self` :
- `static` : `const` `static` `static`
- `struct` :
- `trait` : trait Traits
- `true` : true
- `type` : `type`
- `unsafe` : trait
- `use` : crate `use`
- `where` : Traits `where`
- `while` : `while`

- `!` (`ident!(…)` , `ident!{…}` , `ident![…]`):
- `!` (`!expr`): `Not`
- `!=` (`var != expr`): `PartialEq`
- `%` (`expr % expr`): `Rem`
- `%=` (`var %= expr`):
- `&` (`expr & expr`): `BitAnd`
- `&` (`&expr`):
- `&` (`&type` , `&mut type` , `&'a type` , `&'a mut type`):
- `&=` (`var &= expr`):
- `&&` (`expr && expr`):
- `*` (`expr * expr`): `Mul`
- `*` (`*expr`):
- `*` (`*const type` , `*mut type`):
- `*=` (`var *= expr`):
- `+` (`expr + expr`): `Add`
- `+` (`trait + trait` , `'a + trait`): Traits trait bound
- `+=` (`var += expr`):
- `,` : crate `use`
- `-` (`expr - expr`): `Sub`
- `-` (`- expr`): `Neg`
- `-=` (`var -= expr`):
- `->` (`fn(…) -> type` , `|…| -> type`):
- `-> !` (`fn(…) -> !` , `|…| -> !`):
- `.` (`expr.ident`):
- `..` (`..` , `expr..` , `..expr` , `expr..expr`):
- `..` (`..expr`):
- `..` (`variant(x, ..)` , `struct_type { x, .. }`): “”
- `...` (`expr ... expr`):
- `/` (`expr / expr`): `Div`
- `/=` (`var /= expr`):
- `:` (`pat: type` , `ident: type`):
- `:` (`ident: expr`):
- `:` (`'a: loop {…}`):
- `;` :
- `;` (`[…; len]`):
- `<<` (`expr << expr`): `Shl`
- `<<=` (`var <<= expr`):
- `<` (`expr < expr`): `PartialOrd`
- `<=` (`var <= expr`): `PartialOrd`
- `=` (`var = expr` , `ident = type`): / `type`
- `==` (`var == expr`): `PartialEq`
- `=>` (`pat => expr`):
- `>` (`expr > expr`): `PartialOrd`
- `>=` (`var >= expr`): `PartialOrd`
- `>>` (`expr >> expr`): `Shr`
- `>>=` (`var >>= expr`):

- `@` (`ident @ pat`):
- `^` (`expr ^ expr`): `BitXor`
- `^=` (`var ^= expr`):
- `|` (`expr | expr`): `BitOr`
- `|` (`pat | pat`):
- `|` (`|…| expr`):
- `|=` (`var |= expr`):
- `||` (`expr || expr`):
- `_` : “”

- `'ident` :
- `…u8` , `…i32` , `…f64` , `…usize` , …:
- `"…"` :
- `r"…"` , `r#"…"#` , `r##"…"##` , …:
- `b"…"` : `[u8]`
- `br"…"` , `br#"…"#` , `br##"…"##` , …:
- `'…'` : `char`
- `b'…'` : ASCII
- `|…| expr` :

- `ident::ident` : crate
- `::path` : crate crate `pub use`
- `self::path` : crate `pub use`
- `super::path` : crate `pub use`
- `type::ident` :
- `<type>::…` : `<&T>::…` `<[T]>::…`

- `path<…>` (*e.g.* `Vec<u8>`):
- `path::<…>` , `method::<…>` (*e.g.* `"42".parse::<i32>()`):
- `fn ident<…> …` :
- `struct ident<…> …` :
- `enum ident<…> …` :
- `impl<…> …` :
- `for<…> type` : bound
- `type<ident=type>` (*e.g.* `Iterator<Item=T>`):

- `T: U` : `T` `U` Traits
- `T: 'a` : `T` `'a`
- `'b: 'a` : `'b` `'a`
- `T: ?Sized` :
- `'a + trait` , `trait + trait` : Traits trait bound

- `#[meta]` :
- `#![meta]` :
- `$ident` :

- `$ident:kind` : capture
- `$(…)…` :

- `//` :
- `//!` :
- `///` :
- `/*…*/` :
- `/*!…*/` :
- `/**…*/` :

- `()` :
- `(expr)` :
- `(expr,)` :
- `(type,)` :
- `(expr, …)` :
- `(type, …)` :
- `expr(expr, …)` : `struct enum`
- `ident!(…)` , `ident!{…}` , `ident![…]` :
- `expr.0` , `expr.1` , …:

- `{…}` :
- `Type {…}` : `struct`

- `[…]` :
- `[expr; len]` : `expr len`
- `[type; len]` : `len type`
- `expr[expr]` : `Index IndexMut`
- `expr[..]` , `expr[a..]` , `expr[..b]` , `expr[a..b]` : `Range RangeFrom RangeTo RangeFull` “”

Rust

Rust

- CycloneRegion based memory management in Cyclone
- CycloneSafe manual memory management in Cyclone
- Typeclasses: making ad-hoc polymorphism less ad hoc
- Macros that work together
- Traits: composable units of behavior
- Alias burying -
- External uniqueness is unique enough
- Uniqueness and Reference Immutability for Safe Parallelism
- Region Based Memory Management

- SingularitySingularity: rethinking the software stack
- SingularityLanguage support for fast and reliable message passing in singularity OS
- work stealingScheduling multithreaded computations by work stealing
- Thread scheduling for multiprogramming multiprocessors
- work stealingThe data locality of work stealing
- work stealingDynamic circular work stealing deque - Chase/Lev
- -workhelpWork-first and help-first scheduling policies for async-finish task parallelism - work stealing
- Javafork/joinA Java fork/join calamity - Java fork/joinwork stealing
- Scheduling techniques for concurrent systems
- Contention aware scheduling
- work stealingBalanced work stealing for time-sharing multicores
- Three layer cake
- work stealingNon-blocking steal-half work queues
- ReagentsReagents: expressing and composing fine-grained concurrency
- Algorithms for scalable synchronization of shared-memory multiprocessors

- Crash-only software
- Composing High-Performance Memory Allocators
- Reconsidering Custom Memory Allocation

Rust

- RustGPUGPU programming in Rust
- Parallel closures: a new twist on an old idea - RustNicholas D. Matsakis
- Patina: A Formalization of the Rust Programming LanguageEric Reed
- Experience Report: Developing the Servo Web Browser Engine using RustLars Bergstrom
- Implementing a Generic Radix Trie in RustMichael Sproul
- Reenix: Implementing a Unix-Like Operating System in RustAlex Light
- Evaluation of performance and productivity metrics of potential programming languages in the HPC environmentFlorian WilkensCGoRust
- Nom, a byte oriented, streaming, zero copy, parser combinators library in RustGeoffroy CouprieVLC
- Graph-Based Higher-Order Intermediate RepresentationImpalaRustIR
- Code Refinement of Stencil CodesImpala

> issue

vector

`vector` “” `vector`

slice

`slice` “” `slice`

trait

`trait`

trait object

`trait object` `trait`