



# Let's play with Rust: A Friendly Introduction

@rowdymehul



# Rowdy::About();

- A student
- Lives in Nashik
- Founder & President at Infinite Defense Found
- Independent Researcher
- Specialised in Open Source Security(OSS)
- Mozilla Representative & Resource - Mozilla
- DevOps Engineer & Linux Geek
- Campus Advisory Committee(CAC) – Mozilla
- Rust Mobilizer
- Tech Blogger



# Agenda

1. What is Rust?
2. Why should I use Rust?
3. Install Rust with rustup
4. clippy - the popular Rust static analysis tool
5. Cargo - Rust's awesome package manager
6. rustfmt - the Rust source code formatting tool
7. Play with Rust: A short demo with kits.
8. How to get in touch with the Rust community?
9. Q/A

# What is Rust?

(Baby don't hurt me, don't hurt me, no more)

- Rust is a new systems programming language designed for safety, concurrency, and speed.
- It was originally conceived by Graydon Hoare and is now developed by a team in Mozilla Research and the community.
- Multi-paradigm. Functional, imperative, object-oriented, whenever it makes sense.
- Low-level. Targets the same problem-space as C and C++
- Safe. Lovely, lovely types and pointer lifetimes guard against a lot of errors.

# Why do we need a new system programming language?

- State of art programming language
- Solves a lot of common system programming bugs
- Cargo : Rust Package manager
- Improving your toolkit
- Self learning
- It's FUN ...

# Where can I get it?

- Prebuilt binaries are available at <http://www.rust-lang.org/>
- Source code is available from GitHub <https://github.com/mozilla/rust>
- Kits and resources are available from Rust India GitHub <https://github.com/RustIndia/Rust>

# Rust

- System programming language
- Has great control like C/C++
- Safety and expressive like python





# Best things about Rust

- Strong type system
  - Reduces a lot of common bugs
- Borrowing and Ownership
  - Memory safety
  - Freedom from data races
- Zero Cost abstraction

# Why should I use Rust?

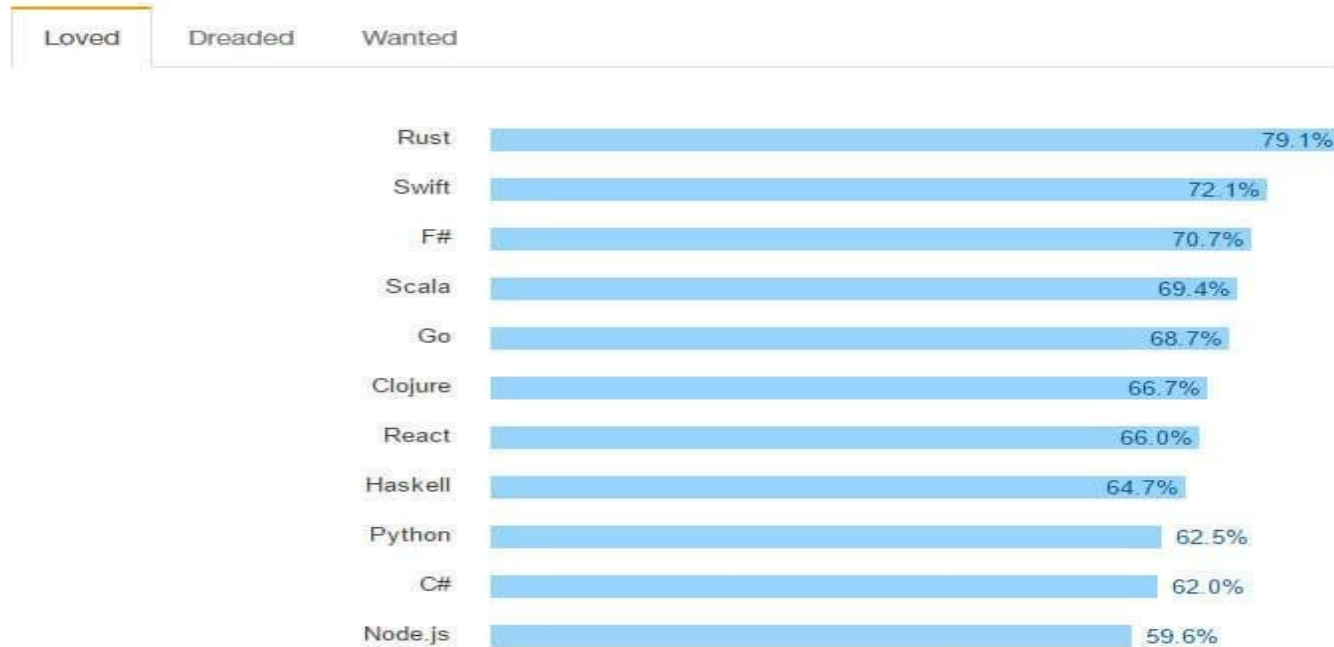
## Own Definition :

Rust is a good choice when you'd choose C++. You can also say, "Rust is a systems programming language that pursuing the trifecta: safe, concurrent, and fast." I would say, Rust is an ownership-oriented programming language.

# Firstly, the reason that I've looked into Rust at first.

- Rust is new enough that you can write useful stuff that would have already existed in other languages
- It gives a relatively familiar tool to the modern C++ developers, but in the much more consistent and reliable ways.
- It is low-level enough that you take account of most resources.
- It's more like C++ and Go, less like Node and Ruby
- cargo is awesome. Managing crates just works as intended, which makes a whole lot of troubles you may have in other languages just vanish with a satisfying *poof*.

According to recent The Stack Overflow survey Rust is the most beloved among developers of all programming languages and frameworks.



% of developers who are developing with the language or tech and have expressed interest in continuing to develop with it

Credits : <https://insights.stackoverflow.com>

# Install Rust with rustup

# Ubuntu / MacOS

- Open your terminal (ctrl + Alt +T)
- `curl -sSf https://static.rust-lang.org/rustup.sh | sh`

```
1) Proceed with installation (default)
2) Customize installation
3) Cancel installation
1
info: updating existing rustup installation

Rust is installed now. Great!
```

# Installing Rust

**rustc** --version

```
mehul@mehul-Inspiron-3542:~$ rustc --version  
rustc 1.11.0 (9b21dcd6a 2016-08-15)
```

**cargo** --version

## # Windows

- Go to <https://win.rustup.rs/>
  - This will download rustup-init.exe
- Double click and start the installation

# Cargo, Rust's Package Manager


Cargo is a tool that allows Rust projects to declare their various dependencies and ensure that you'll always get a repeatable build.

To accomplish this goal, Cargo does four things:

- Introduces two metadata files with various bits of project information.
- Fetches and builds your project's dependencies.
- Invokes rustc or another build tool with the correct parameters to build your project.
- Introduces conventions to make working with Rust projects easier.

[Creating a new project](#)

<https://crates.io/>



**crates.io**  
Rust Package Registry

🔍 Click or press 'S' to search...

[Browse All Crates](#) | [Docs](#) ▾ | [Log in with GitHub](#)

# The Rust community's crate registry

[📦 Install Cargo](#)

[🚩 Getting Started](#)

---

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

📥

**245,040,023** Downloads

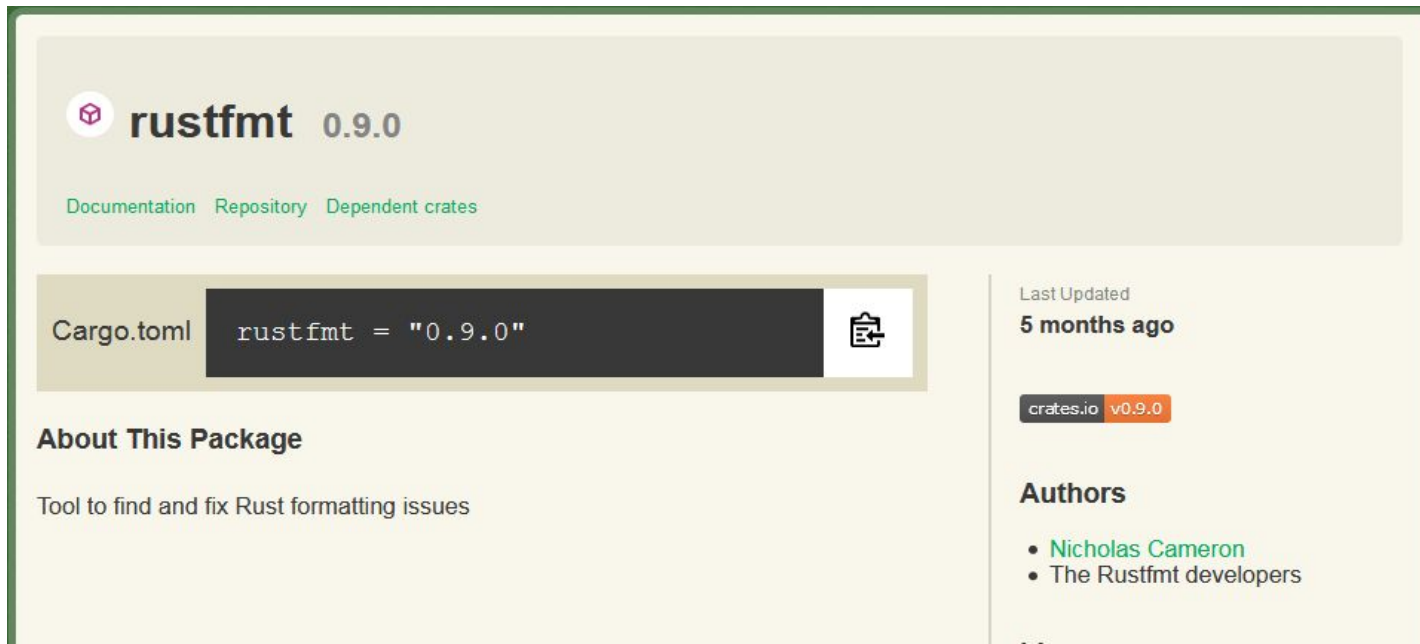
📦

**12,108** Crates in stock



# rustfmt

- A tool for formatting Rust code according to style guidelines.



The screenshot shows the crates.io page for the `rustfmt` crate, version 0.9.0. The page has a light beige background with a dark green border. At the top, the crate name `rustfmt` is displayed in a large, bold, dark font, followed by the version number `0.9.0`. Below this, there are three links: [Documentation](#), [Repository](#), and [Dependent crates](#), all in a green font. A section titled "Cargo.toml" shows a code snippet: `rustfmt = "0.9.0"`. To the right of the code is a clipboard icon. Below the code snippet, the text "About This Package" is followed by the description: "Tool to find and fix Rust formatting issues". On the right side of the page, there is a section titled "Last Updated" with the text "5 months ago". Below this is a badge that says "crates.io v0.9.0". Further down, there is a section titled "Authors" with a list of authors: [Nicholas Cameron](#) and "The Rustfmt developers".

**rustfmt 0.9.0**

[Documentation](#) [Repository](#) [Dependent crates](#)

Cargo.toml `rustfmt = "0.9.0"`

**About This Package**

Tool to find and fix Rust formatting issues

Last Updated  
**5 months ago**

crates.io v0.9.0

**Authors**

- [Nicholas Cameron](#)
- The Rustfmt developers

# Rust Playground

- A web interface for running Rust code.
- The interface can also be accessed in most Rust-related channels on [irc.mozilla.org](https://irc.mozilla.org).
- To use Playbot in a public channel, address your message to it.
  - `<you> playbot: println!("Hello, World");`  
`-playbot: #rust-offtopic- Hello, World`  
`-playbot: #rust-offtopic- ()`  
`<you> playbot: 1+2+3`  
`-playbot: #rust-offtopic- 6`
- You can also private message Playbot your code to have it evaluated. In a private message, don't preface the code with playbot's nickname:
  - `/msg playbot println!("Hello, World");`

Let's play : <https://play.rust-lang.org/>

The screenshot shows the Rust Playground web interface. The browser's address bar displays `https://play.rust-lang.org`. The interface includes a toolbar with various options: a 'Run' button, 'Tools' (ASM, LLVM IR, MIR, Format, Clippy, Share), 'Mode' (Debug, Release), 'Channel' (Stable, Beta, Nightly), and 'Config'. The code editor on the left contains the following Rust code:

```
1 fn main() {  
2     println!("Hello, world!");  
3 }
```

On the right, the 'Execution' panel shows the output of the program. It includes a 'Standard Error' section with the following text:

```
Compiling playground v0.0.1 (file:///playground)  
Finished dev [unoptimized + debuginfo] target(s) in 0.45 secs  
Running `target/debug/playground`
```

Below this, the 'Standard Output' section displays the result of the program's execution:

```
Hello, world!
```

# Type System

# The traditional Hello World

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



# A bit complex example

```
fn avg(list: &[f64]) -> f64 {
```

```
    let mut total = 0;
```

```
    for el in list{
```

```
        total += *el;
```

```
    }
```

```
    total/list.len() as f64
```

```
}
```

# HLL version

```
fn avg(list: &[f64]) -> f64 {  
    list.iter().sum::<f64>() / list.len() as f64  
}
```

# Parallel Version (Rayon)

```
fn avg(list: &[f64]) -> f64 {  
    list.par_iter().sum::<f64>() / list.len() as f64  
}
```



# Fold

```
fn avg(list: &[f64]) -> f64 {  
    list.par_iter().fold(0., |a,b| a + b) / list.len() as f64  
}
```

# Primitive Types

# bool

```
let bool_val: bool = true;
```

```
println!("Bool value is {}", bool_val);
```

# char

```
let x_char: char = 'a';
```

```
// Printing the character
```

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

# i8/i16/i32/i64/usize

```
let num =10;
```

```
println!("Num is {}", num);
```

```
let age: i32 =40;
```

```
println!("Age is {}", age);
```

```
println!("Max i32 {}",i32::MAX);
```

```
println!("Max i32 {}",i32::MIN);
```

# Other Primitive Types

- `u8/u16/u32/u64/usize`
- `f32/f64`

# Tuples

```
// Declaring a tuple
```

```
let rand_tuple = ("Mozilla Science Lab", 2016);
```

```
let rand_tuple2 : (&str, i8) = ("Viki",4);
```

```
// tuple operations
```

```
println!(" Name : {}", rand_tuple2.0);
```

```
println!(" Lucky no : {}", rand_tuple2.1);
```

# Arrays

```
let rand_array = [1,2,3]; // Defining an array
```

```
println!("random array {:?}",rand_array );
```

```
println!("random array 1st element {}",rand_array[0] ); // indexing starts with 0
```

```
println!("random array length {}",rand_array.len() );
```

```
println!("random array {:?}",&rand_array[1..3] ); // last two elements
```



# String

```
let rand_string = "I love Mozilla Science <3"; // declaring a random string
```

```
println!("length of the string is {}",rand_string.len() ); // printing the length of the string
```

```
let (first,second) = rand_string.split_at(7); // Splits in string
```

```
let count = rand_string.chars().count(); // Count using iterator count
```

# Complex Data structures

# struct

// define your custom user datatype

**struct** Circle {

    x : f64,

    radius : f64,

}

# Rust “Class”

```
impl Circle {
```

```
// pub makes this function public which makes it accessible outside the scope {}
```

```
    pub fn get_x(&self) -> f64 {
```

```
        self.x
```

```
    }
```

```
}
```

# Traits

- Interfaces
- Operator overloading
- Indicators of behaviour
- Bounds for generic
- Dynamic dispatch

# Trait Sample

```
// create a functionality for the datatypes
```

```
trait HasArea {
```

```
    fn area(&self) -> f64;
```

```
}
```

```
// implement area for circle
```

```
impl HasArea for Circle {
```

```
    fn area(&self) -> f64 {
```

```
        3.14 * (self.r *self.r)
```

```
    }
```

```
}
```

# Ownership

In Rust, every value has an “owning scope,” and passing or returning a value means transferring ownership (“moving” it) to a new scope

```
fn make_vec() {  
    let mut vec = Vec::new(); // owned by make_vec's scope  
    vec.push(0);  
    vec.push(1);  
    // scope ends, `vec` is destroyed  
}
```

# Example 1

```
fn foo{
```

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

```
    let x = v;
```

```
    println!("{:?}",v); // ERROR : use of moved value: "v"
```

```
}
```



## Ownership - Ex 2

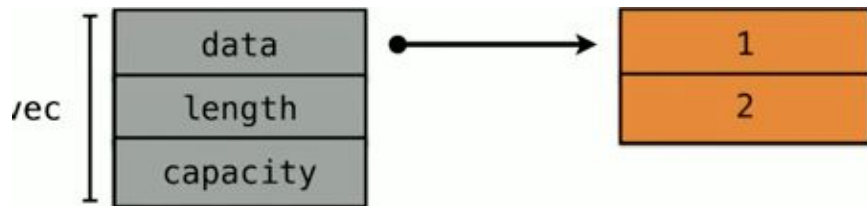
```
fn make_vec() -> Vec<i32> {  
    let mut vec = Vec::new();  
    vec.push(0);  
    vec.push(1);  
    vec // transfer ownership to the caller  
}  
  
fn print_vec(vec: Vec<i32>) {  
    // the `vec` parameter is part of this scope, so it's owned by `print_vec`  
  
    for i in vec.iter() {  
        println!("{}", i)  
    }  
  
    // now, `vec` is deallocated  
}  
  
fn use_vec() {  
    let vec = make_vec(); // take ownership of the vector  
    print_vec(vec);       // pass ownership to `print_vec`  
}
```

```
fn print(v : Vec<u32>) {  
    println!("{:?}", v);  
}
```

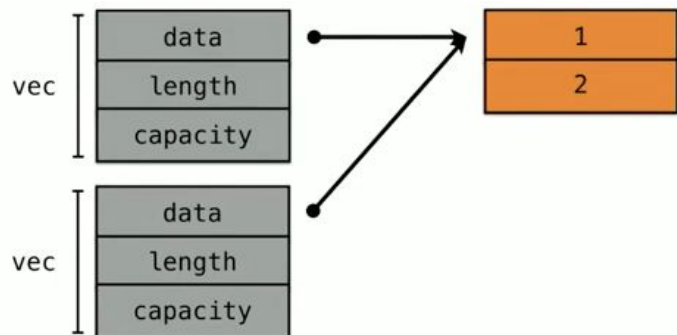
```
fn make_vec() {  
    let v = vec![1,2,3];  
    print(v);  
    print(v); // ERROR : use of moved value: "v"  
}
```

# Ownership

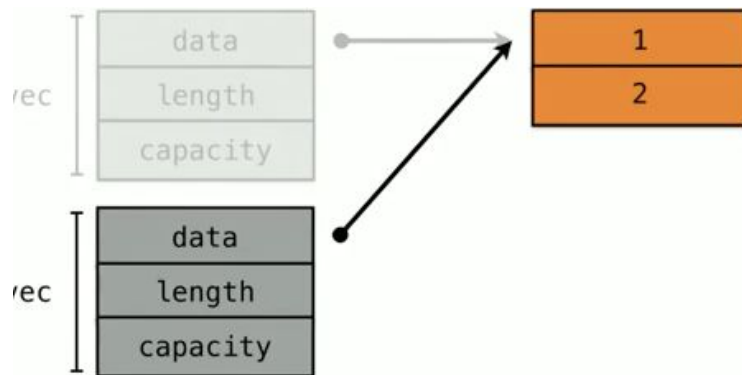
Step 1.



Step 2.



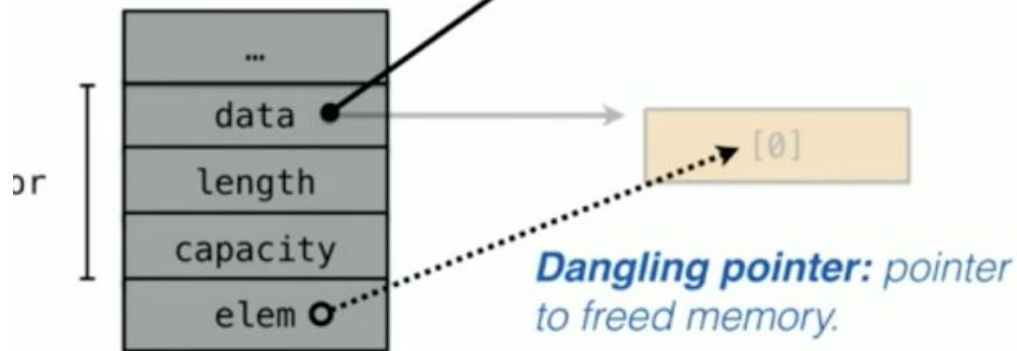
Step 3.



# Aliasing

More than one pointer to the same memory

```
auto& elem = vector[0];  
vector.push_back(some_string);  
cout << elem;
```



Ownership concepts avoids Aliasing

# Borrowing

If you have access to a value in Rust, you can lend out that access to the functions you call

```
fn print_vec(vec: &Vec<i32>) {  
    // the `vec` parameter is borrowed for this scope  
  
    for i in vec.iter() {  
        println!("{}", i)  
    }  
  
    // now, the borrow ends  
}  
  
fn use_vec() {  
    let vec = make_vec(); // take ownership of the vector  
    print_vec(&vec);       // lend access to `print_vec`  
    for i in vec.iter() { // continue using `vec`  
        println!("{}", i * 2)  
    }  
    // vec is destroyed here  
}
```

# Types of Borrowing

There is two type of borrowing in Rust, both the cases aliasing and mutation do not happen simultaneously

- Shared Borrowing (&T)
- Mutable Borrow (&mut T)

# &mut T

```
fn add_one(v: &mut Vec<u32> ) {
```

```
    v.push(1)
```

```
}
```

```
fn foo() {
```

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

```
    add_one(&mut v);
```

```
}
```

# Rules of Borrowing

- Mutable borrows are exclusive
- Cannot outlive the object being borrowed



# Cannot outlive the object being borrowed

```
fn foo{
```

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

```
let borrow1 = &v;
```

```
let borrow2 = &v;
```

```
add_one(&mut v): // ERROR : cannot borrow 'v' as mutable because
```

}

it is also borrowed as immutable

# Lifetimes

```
let outer;
```

```
{
```

```
    let v = 1;
```

```
    outer = &v; // ERROR: 'v' doesn't live long
```

```
}
```

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

# Rain Of Rust - A Global Rust Campaign



**#rainofrust**

**moz://a**

# Key stats of RainOfRust campaign June 2017:

- 21 offline events across 10 regions globally. Ref
- 4 online meeting which is recorded in Air Mozilla. Ref
- As part of the campaign the RainOfRust has Rust teaching kits which currently has 3 application-oriented activities
- The Github repo has received more than 500+ views in the within a week,

[Read more](#)

# Rain Of Rust - Glimpses



#rainofrust



# Rain Of Rust - Glimpses



# **Let's Meet Friends of Rust!!**

**(Organizations running Rust in production)**

# Getting started with Rust community

- Follow all the latest news at Reddit Channel
  - <https://www.reddit.com/r/rust/>
- Have doubts, post in
  - <https://users.rust-lang.org>
  - #rust IRC channel
- Want to publish a crate,
  - <https://crates.io>
- Follow @rustlang in twitter,
  - <https://twitter.com/rustlang>
- Subscribe to <https://this-week-in-rust.org/> newsletter



# Getting started with Rust community

- Create your rustaceans profile,
  - Fork <https://github.com/nrc/rustaceans.org>
  - Create a file in data directory with <github\_id>.json
    - Ex: rowdymehul.json

```
{  
  "name": "Mehul Patel",  
  "irc": "rowdymehul",  
  "irc_channels": ["rust"],  
  "show_avatar": true,  
  "email": "iamrowdymehul@gmail.com",  
  "discourse": "rowdymehul",  
  "reddit": "rowdymehul",  
  "twitter": "@rowdymehul",  
  "blog": "https://rowdymehul.wordpress.com/",  
  "notes": "OpenSource Enthusiast"  
}
```

---

**Adopt Rust today !!**

# Contribute & Join Rust India!



Rust India

We are Rust India Community.

Repositories 2

People 3

Teams 0

Projects 0

Settings

Search repositories...

Type: All

Language: All

Customize pinned repositories

New

## volunteer-contributions

Repo for the volunteers to report their contributions and share their experience/reports

rust

contributions

mozilla

reports

Updated a day ago

## Top languages

Rust

## Most used topics

Manage

mozilla

rust

## Rust

Rust India Community Repository

rust

mozilla

rustlang

rust-community

mozillaindia

Rust

★ 18

🔗 18

MIT

Updated 2 days ago

## People

3 >



prathameshchavan

Prathamesh Chavan



rowdymehul

Mehul Patel

- <https://github.com/RustIndia/>
- <https://github.com/RustIndia/Rust>
- <https://github.com/servo>

# **Rust and the Future of Systems Programming**





**Presentation  
Finished.**

**Any questions?**

# References

- Segfault: <http://stackoverflow.com/questions/2346806/what-is-a-segmentation-fault>
- BufferOverflow: <http://stackoverflow.com/questions/574159/what-is-a-buffer-overflow-and-how-do-i-cause-one>
- Rust Website: <https://www.rust-lang.org/en-US/>
- Community Forum: <https://users.rust-lang.org/>
- Rust Book: <https://doc.rust-lang.org/book/>
- Why should I use Rust? <https://medium.com/@rowdymehul/31bc292923da>
- Unraveling Rust Design: <https://dvigneshwer.wordpress.com/2017/02/25/unraveling-rust-design/>

# Thank You

- Tweet at #RustIndia #RustLang
- Join [RustIndia Telegram group](#)