

WRITE YOURSELF A SHELL IN RUST

A **6-hour course** to learn
intermediate Rust
concepts



ABOUT THE TEAM

Matthias Endler

- Rust consultant at corrode
- Started with Rust in 2015
- Member of the Rust Cologne Meetup
- Hosted [Hello Rust](#) YouTube channel

Marco Ieni

- Rust Backend Engineer at Prisma
- Host of [RustShip](#) podcast
- Author of [release-plz](#), automating Rust package publication

ABOUT THE WORKSHOP

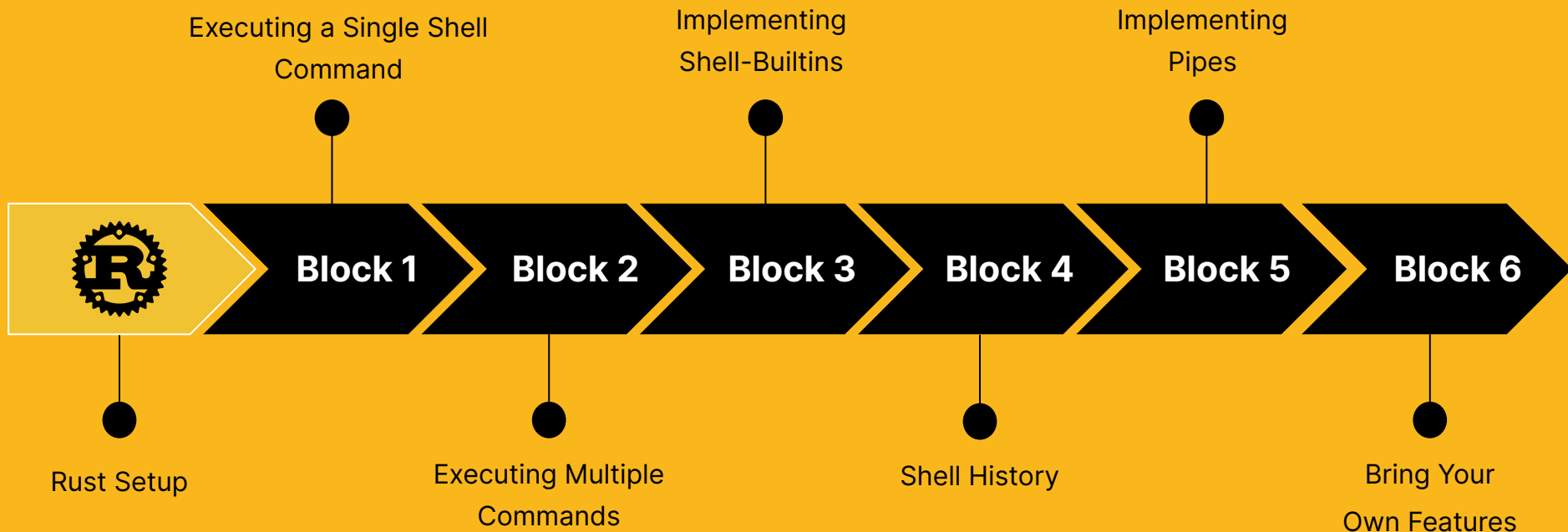
Goals

- Learn intermediate Rust
- Work on a real-world project
- Use plain Rust; no dependencies
- Focus on idiomatic code

Structure

- 6 hours total
- Split up into six blocks
- Roughly one hour per block

SCHEDULE



BLOCK 0 - RUST SETUP

Main objective

- Install Rust using [rustup](#) or any other way.
- Run `rustc -V` to see if everything is okay.

Bonus track

- Install clippy
`rustup component add clippy`
- Set up your project with additional clippy lints.
([Example setup](#))
- Set up rust analyzer for code completion
(<https://rust-analyzer.github.io/>)

BLOCK 1 - EXECUTING A SINGLE SHELL COMMAND

Main objectives

- Write a shell which can run a single command on a separate process.
- *Hint:* Look for APIs in the standard library to do that.
- Print the output to stdout.

Bonus track

- Create a `Cmd` struct holding the binary and the arguments.
- Make the code as idiomatic as possible.
(`cargo clippy` should run without errors.)
- Write some unit tests to make sure that command parsing works.

BLOCK 1 - PROJECT STRUCTURE

```
fn main() {  
    loop {  
        // Read line from standard input  
        // "Parse" line into executable command  
        // Execute the command in a separate process  
        // Show output  
    }  
}
```

BLOCK 1 - REPOSITORY

<https://github.com/corrode/rush>

BLOCK 2 - EXECUTING MULTIPLE COMMANDS

Main objectives

- Try to run two or more commands separated by ; in sequence.
- Print all output in sequence to stdout.

Bonus track

- Implement && and | |
- Write an integration test for chaining multiple commands.
(echo 'hello' && echo 'world')

BLOCK 2 - EXECUTING MULTIPLE COMMANDS

Main objectives

```
> echo 1; echo 2  
1  
2
```

Bonus track

```
> true && echo "output"  
output  
> false && echo "output"  
>
```

```
> true || echo "output"  
> false || echo "output"  
output  
>
```

BLOCK 3 - IMPLEMENTING SHELL-BUILTINS

Main objectives

- Implement the ``cd`` shell builtin.
- Implement the ``exit`` shell builtin.

Bonus track

- Implement ``exec`` builtin

BLOCK 3 - IMPLEMENTING SHELL-BUILTINS (EXAMPLES)

Main objectives

```
> pwd
/dir1
> cd /dir2
> pwd
/dir2
```

```
> exit
(Shell gets closed)
```

Bonus track

```
> exec fish
Welcome to fish, the friendly
interactive shell
```

BLOCK 4 - IMPLEMENT SHELL HISTORY

Main objectives

- Store all executed commands in the shell history.
- Add a `history` builtin, which lists the commands

Bonus track

- Come up with a solid storage format (e.g. adding metadata)
- Add support for alternative storage options (e.g. `sqlite`)

BLOCK 4 - IMPLEMENT SHELL HISTORY

Main objectives

```
> history  
echo 1  
echo 2
```

Bonus track

```
in sqlite:  
> SELECT * from history
```

BLOCK 5 - IMPLEMENTING PIPES

Main objectives

- Implement pipes, which are a way to feed the output of one command into another one.

Syntax:

```
command1 | command2
```

Bonus track

- Support multiple pipes:
`c1 | c2 | c3`
- Add redirection:
`c1 > output.txt`
- Think about ways to make command representation more idiomatic.

BLOCK 5 - IMPLEMENTING PIPES (EXAMPLES)

Main objectives

```
> echo foo | grep -c foo  
1
```

Bonus track

```
> ps auxwww | grep fred | more
```

```
> echo 1 > test.txt
```


BLOCK 6 - BRING YOUR OWN FEATURES!

Main objectives

- It's all free-style from here.
What will you do next?
 - Readline support
 - Control signals
 - Command completion
 - use a grammar for parsing
 - Implement more shell-builtins
 - Surprise us! (Optional
show-and-tell at the end)

Bonus track

- Get inspired by looking at existing shells:
 - [ion](#) (Rust)
 - [elvish](#) (Go)
 - ["the other rush"](#) (Rust)

SHOW AND TELL

An illustration on a yellow background showing a mechanical device. A yellow container is tilted, pouring a thick, white, viscous liquid into a circular base. The base contains a large yellow gear with a black silhouette of a person inside it. The entire scene is rendered in a stylized, mechanical, and industrial aesthetic.

WHAT WE LEARNED

- Intermediate Rust concepts
 - Command handling
 - Parsing
 - Piping inputs
- Idiomatic code
 - Command abstractions
 - Unit tests
- Zero dependencies
 - Plain Rust is powerful
 - No magic