Write your own shell in Rust!

Santiago Pastorino • Matthias Endler
Special Guest: Alex Crichton

Rust Belt Rust Workshop 2018

About us

Santiago Pastorino

- WyeWorks co-founder
- Ruby on Rails core team alumni
- Started with Rust in 2017
- Member of Rust compiler NLL WG
- Rust Latam conference organizer
- Rust Montevideo Meetup organizer

Matthias Endler

- Started with Rust in 2015
- Member of the Rust Cologne Meetup
- Runs *Hello Rust* YouTube channel

About the Workshop

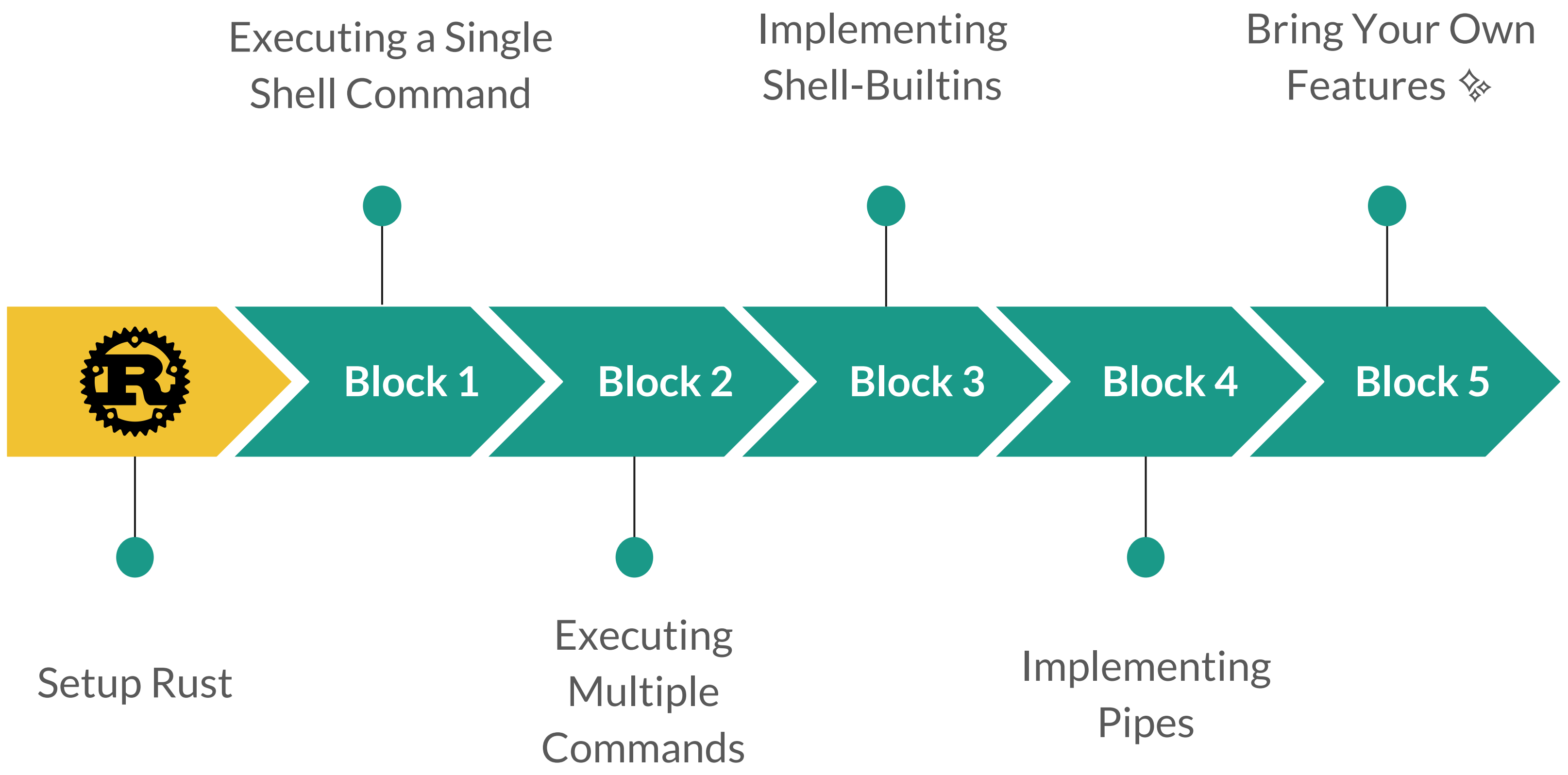
Goals

- Learn some Rust
- Intermediate concepts
- Work on your first Rust project

Structure

- 6 hours total
- split up into five blocks
- roughly one hour per block

Schedule


Executing a Single Shell Command
Implementing Shell-Builtins
Bring Your Own Features ✨
Block 1
Block 2
Block 3
Block 4
Block 5
Setup Rust
Executing Multiple Commands
Implementing Pipes

Block 0 - Check your (nightly) Rust installation

Main objective

- Install nightly Rust using rustup or any other way.
- Run `rustc -V` to see if you're golden.

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

- Write some unit tests to make sure that the shell works.
- Make the code as idiomatic as possible.

Block 1 - Pseudo Code

```
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://gitlab.com/mre_/rush

mre_

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.

Share solutions here: **http://tiny.cc/rustlang2**
Repository: https://gitlab.com/mre_/rush

tiny.cc/rust-slides

Block 2 - Executing Multiple Commands (Examples)

Main objectives

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

Bonus track

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

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

Share solutions here: **http://tiny.cc/rustlang2**
Repository: https://gitlab.com/mre_/rush

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
```

Share solutions here: **http://tiny.cc/rustlang2**
Repository: https://gitlab.com/mre_/rush

Block 4 - 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 4 - Implementing Pipes (Examples)

Main objectives

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

Bonus track

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

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

Block 5 - 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!

Bonus track

- Get ✨ *inspired* ✨ by looking at existing shells, e.g.
    - ion (Rust)
    - elvish (Go)
    - “the *other* rush” (Rust)