How to make command line apps in rust

by Łukasz Biel

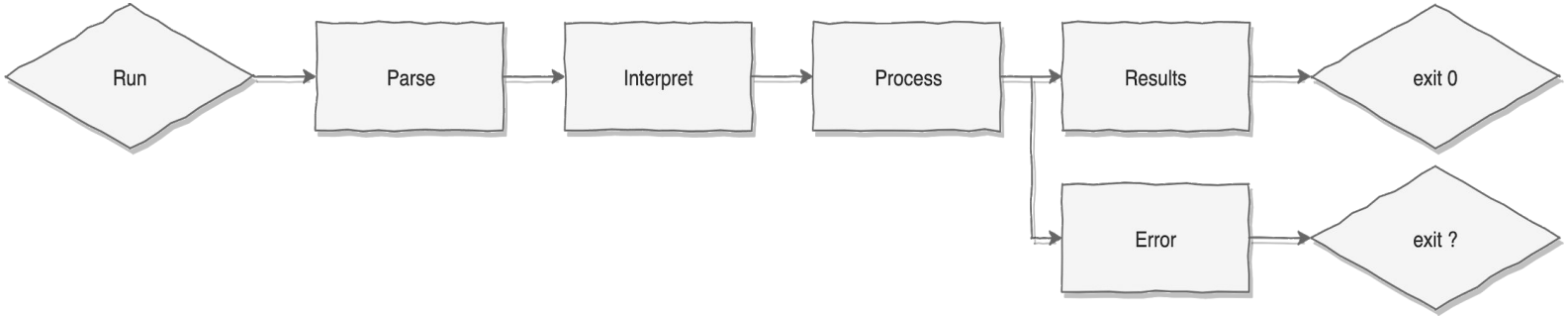
Table of contents

Who am I?

- developer at Anixe since 2017, working with Rust
- hobbyist game dev
- I help with organization of 2 game dev oriented hackatons in Wrocław: **Sensei Game Jam** and **TK Game Jam**

https://luke-biel.github.io/ & https://twitter.com/dot_luke_biel

CLI app structure



Parsing command line arguments

- by hand (`std::env::args()`)
- structopt (wrapper over clap)
- docopt
- clap

Process & Interpret - honorable mentions

- reqwest - simple API clients
- serde - to serialize/deserialize any data structure (configs, REST API responses, etc)
- shellfn - proof of concept shell interface
- lazy_static - statically living objects located in heap memory
- regex - quite self-explanatory
- dirs - crate that provides platform-specific user directories
- rustyline - repl support

Logging

- println!
- log
- slog

…

- panic!

Error handling

- custom errors
- failure
- error-chain

Tips

- always provide readme, cargo-readme is very useful crate for that purpose
- keep your --help up to date
- split between lib and bin targets
- test!

Case study - bts

```rust
// main.rs
fn main() {
    let args: Args = Args::from_args();


    let result = match args.command {
        Command::New(spawn_args) => new(spawn_args, args.config_location),
        Command::Register(register_args) => register(register_args, args.config_location),
    };


    if let Err(e) = result {
        println!("Critical error occurred!\n{:?}", e);
    }
}
```

Case study - bts

```rust
// lib.rs
pub fn new<P: AsRef<Path>>(args: NewArgs, config_location: P) -> Result<(), Error> {
    …
}

pub fn register<P: AsRef<Path>>(args: RegisterArgs, config_location: P) -> Result<(), Error> {
    …
}
```

Case study - bts

```rust
// args.rs
#[derive(StructOpt)]
#[structopt(name = "bts", about = "Automatic template file generator.")]
/// Generate file snippets at will
pub struct Args {
   /// Location of snippets storage
   #[structopt(env = "BT_HOME", default_value = Self::default_template_folder())]
   pub config_location: PathBuf,
   #[structopt(flatten)]
   pub command: Command,
}
```

Case study - bts

```
// error.rs
#[derive(Debug)]
pub enum Error {
   Other(Box<dyn Debug>),
   CopyError(io::Error),
   Lookup(io::Error),
}
```

Case study - bts

```
// acceptance.rs
#[test_case(false, "file_example", "file_example", 1; "single file is copied to destination")]
fn acceptance(with_parent: bool, template_name: &str, target_name: &str, max_depth: u8) {
    ...
}
```

Live coding

… let’s proceed to an editor

complete project can be found @ https://github.com/luke-biel/wiki-meetup-client

Code I wrote would be one-liner in bash

Q&A