Speaker Bio

Cristian Vlasceanu

- https://www.linkedin.com/in/cristianvlasceanu/
- Used C/C++ professionally for nearly 30 years
  - Amazon
  - Microsoft
  - Tableau Software / Salesforce

My First Rust Project as a Long Time C++ Dev

First impressions and commentary

Introduction

Unix commands are (still) an expedient way for:

- Grepping for pattern in log files
- Looking at config files
- Checking available disk space
- Inspecting running processes
- Searching for files

On Windows …

- Cygwin: a good (older) solution, but
  - May take some time to install, large

- WSL2: a good (newer) solution, but
  - Maps user home elsewhere than the native environment
  - Slower filesystem than native
  - Runs in its own VM - ‘ps’ command shows processes in VM, not native

On Windows … (cont)

- Write my own command interpreter (shell):
  - Excuse to dive into Rust
  - Light-weight, built-in common commands (cat, cp, diff, ls)
  - Full control / ownership of code, customize to own needs
  - Consistent behavior across different platforms and systems

- Project at: https://github.com/cristivlas/shmy

First Speed Bumps

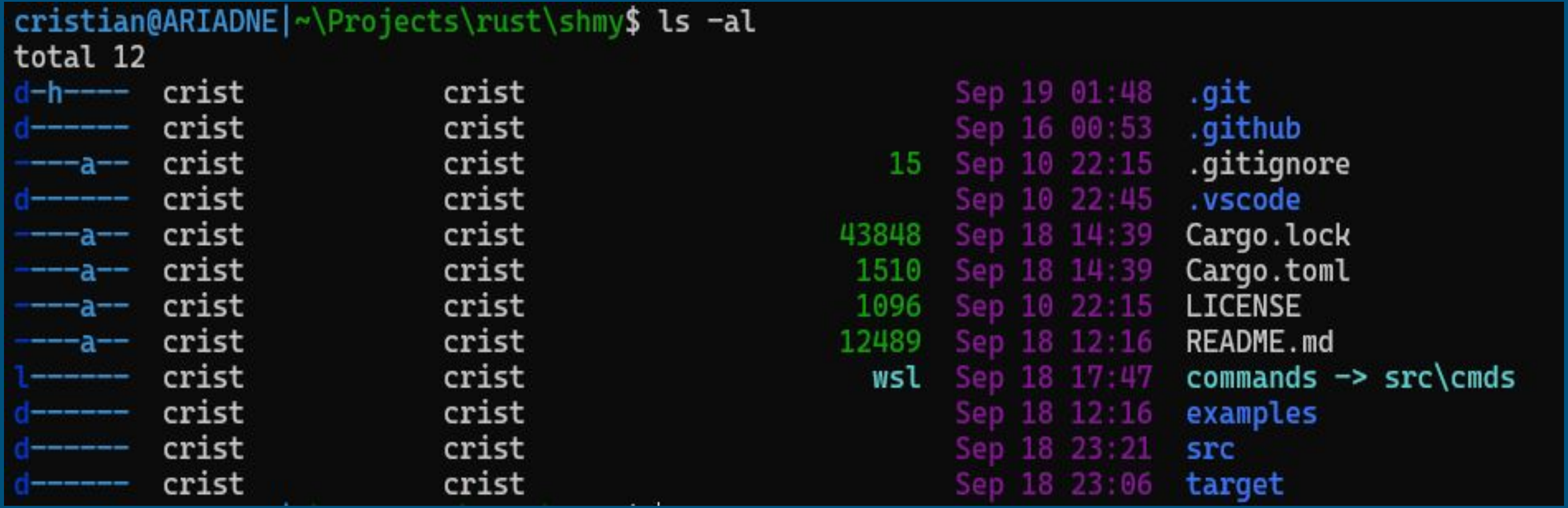
- Rc / Arc are smart pointers like std::shared_ptr
    - **without the shared part**!
    - std::cell::RefCell feels hacky (lie about immutability?)

- It is more tempting for a beginner to over-allocate memory (clone) than to annotate lifetimes and understand std::borrow::Cow

Detour (Back to WSL)

- Symbolic links created under WSL cannot be opened by cmd.exe, File Explorer, etc.
- Not “seen” by Rust (std::fs, std::path::PathBuf::canonicalize) either
- WSL symbolic links are implemented as NTFS Reparse Points
  - https://en.wikipedia.org/wiki/NTFS_reparse_point

Solution using Windows Crate



```
cristian@ARIADNE|~\Projects\rust\shmy$ ls -al
total 12
d-h----   crist             crist                           Sep 19 01:48  .git
d------   crist             crist                           Sep 16 00:53  .github
----a--   crist             crist                        15 Sep 10 22:15  .gitignore
d------   crist             crist                           Sep 10 22:45  .vscode
----a--   crist             crist                     43848 Sep 18 14:39  Cargo.lock
----a--   crist             crist                      1510 Sep 18 14:39  Cargo.toml
----a--   crist             crist                      1096 Sep 10 22:15  LICENSE
----a--   crist             crist                     12489 Sep 18 12:16  README.md
l------   crist             crist                       wsl Sep 18 17:47  commands -> src\cmds
d------   crist             crist                           Sep 18 12:16  examples
d------   crist             crist                           Sep 18 23:21  src
d------   crist             crist                           Sep 18 23:06  target
```

Feeling Unsafe

```rust
// Retrieve the reparse point data
unsafe {
    DeviceIoControl(
        HANDLE(file.as_raw_handle()),
        FSCTL_GET_REPARSE_POINT,
        None,
        0,
        Some(buffer.as_mut_ptr() as *mut _),
        buffer.len() as u32,
        Some(&mut bytes_returned),
        None,
    )
 }
 .map_err(|_| io::Error::last_os_error())?;
```

Impressions So Far

- Safety features oversold?
  - Wrapping legacy (Windows) APIs feels like ATL
- Memory mapped files are considered unsafe?
  - Came as a surprise, common with large datasets
- Allocation failures panic, no std::bad_alloc
  - It’s okay, we’ve got memory
  - Need to think carefully about bad user inputs!
- Love the Cargo Ecosystem
  - I do not miss: building boost (jam anyone?), make, cmake, ninja and all that junk
  - Ease of writing unit tests and generating documentation

Impressions… (cont)

Also do not miss:

- C/C++ Lib Artifact built with g++ under one Linux distro mixed with clang under different distro
- Rust / cargo dependency management avoids such problems

Bonus: Know Your Test Environment

```
cristian@ARIADNE|~\Projects\rust\shmy$ if 1 (True) else (False)

True
```

- Failed in github on Windows (but passed under MacOS and Linux!)
- Passed locally 100%
- “Too simple to mock”

Explanation: Commands vs String Literals

```
cristian@ARIADNE|~\Projects\rust\shmy$ bogus -al
1:6 Cannot subtract strings
cristian@ARIADNE|~\Projects\rust\shmy$ ls -al
```

This works! “-al” **interpreted as command argument**, not rhs of subtract expression

The if (1) (True) test failed because:

“True” and “False” exist as commands in the github Windows VMs