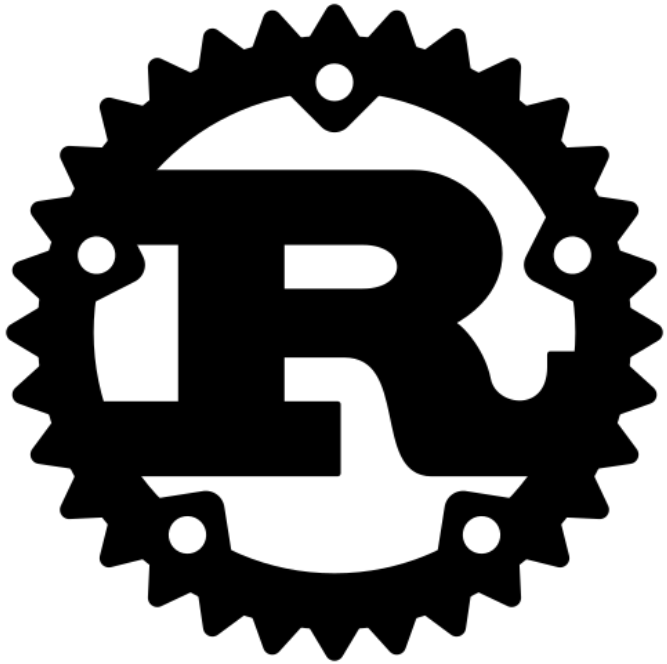
How do Rust applications get into Debian?



debian

whoami

- Matthias Geiger (werdahias)
- Debian user since 2017, contributor since 11/2021, maintainer since 12/2022
- packaging mostly rust applications / libraries for Debian
- currently studying Electrical Engineering and Information Technology
- interests: hiking, swimming, reading, heavy metal

What is Rust? — Quick terminology

- system programming language
- memory safe by default
- `rustc`: compiler

What is Rust? — Quick terminology

- system programming language
- memory safe by default
- `rustc`: compiler
- `cargo`: build system and package manager
- uses crates to build the binary

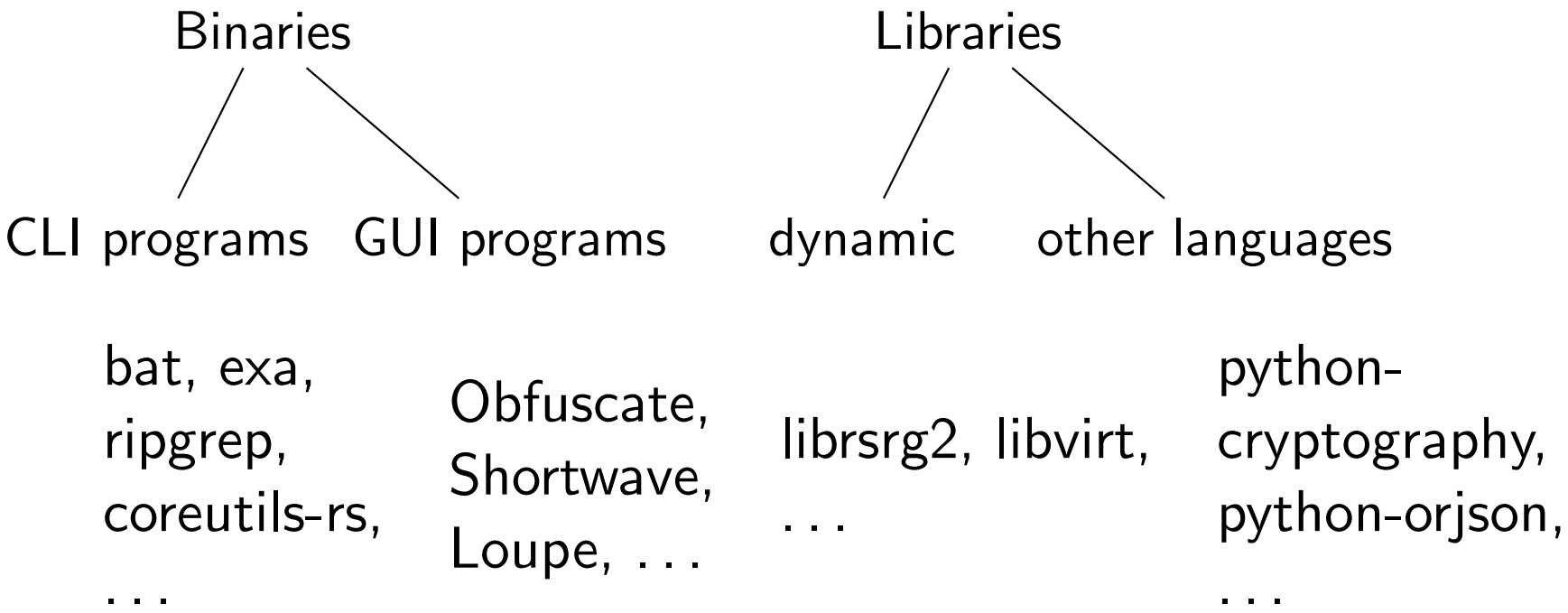
What is Rust? — Quick terminology

- system programming language
- memory safe by default
- `rustc`: compiler
- `cargo`: build system and package manager
- uses crates to build the binary
- crates: equivalent to C/C++ libraries
- distributed by upstream/s at a central registry, crates.io

What is Rust? — Quick terminology

- system programming language
- memory safe by default
- `rustc`: compiler
- `cargo`: build system and package manager
- uses crates to build the binary
- crates: equivalent to C/C++ libraries
- distributed by upstream/s at a central registry, crates.io
- `debcargo`: Debian-internal tool to do heavy lifting
- creates a debianized source dir, translates dependencies etc.

Projects written in Rust



How does the packaging work? — The Debian Rust teams' workflow

- maintaining `rustc`, `cargo`, `debcargo`, `rustup`

How does the packaging work? — The Debian Rust teams' workflow

- maintaining `rustc`, `cargo`, `debcargo`, `rustup`
- big monorepo for crates and binary crates
- maintaining crates and packaging new ones
- updates, big transitions, bug triaging, package removal

How does the packaging work? — The Debian Rust teams' workflow I

- determining dependency trees and missing dependencies
- → `cargo-debstatus`

How does the packaging work? — The Debian Rust teams' workflow II

```
04:27:57 werdahias@yggsdrasil magic-wormhole.rs ±|master ✗|→ cargo debstatus
magic-wormhole v0.6.0 (/home/werdahias/magic-wormhole.rs)
├── async-io v1.13.0 (in debian)
├── async-std v1.12.0 (in debian)
├── async-tar v0.4.2 (in debian)
├── async-trait v0.1.68 (in debian)
├── async-tungstenite v0.23.0 (in debian NEW queue)
├── base64 v0.21.2 (in debian)
├── bytecodec v0.4.15 (in debian)
├── derive_more v0.99.17
│   ├── proc-macro2 v1.0.60 (in debian)
│   ├── quote v1.0.28 (in debian)
│   └── syn v1.0.109 (in debian)
├── futures v0.3.28 (in debian)
├── futures_ringbuf v0.4.0
│   ├── futures v0.3.28 (in debian)
│   ├── log v0.4.19 (in debian)
│   └── ringbuf v0.3.3 (in debian NEW queue)
│   [build-dependencies]
│   └── rustc_version v0.4.0 (in debian)
├── hex v0.4.3 (in debian)
├── hkdf v0.12.3 (in debian)
├── if-addrs v0.10.1 (in debian)
├── instant v0.1.12 (in debian)
├── libc v0.2.146 (in debian)
├── log v0.4.19 (in debian)
├── mockall_double v0.3.0
│   ├── cfg-if v1.0.0 (in debian)
│   ├── proc-macro2 v1.0.60 (in debian)
│   ├── quote v1.0.28 (in debian)
│   └── syn v1.0.109 (in debian)
├── noise-protocol v0.1.4 (in debian NEW queue)
├── percent-encoding v2.3.0 (in debian)
├── rand v0.8.5 (in debian)
├── rmp-serde v1.1.1 (in debian)
├── serde v1.0.164 (in debian)
├── serde_derive v1.0.164 (in debian)
├── serde_json v1.0.97 (in debian)
├── sha-1 v0.10.1
│   ├── cfg-if v1.0.0 (in debian)
│   ├── cpufeatures v0.2.8
```

How does the packaging work? — The Debian Rust teams' workflow I

- packaging a crate: two simple scripts utilising `debcargo`
- minimal tweaking by hand: `copyright` and `debcargo.toml`
- optionally: patches, `debian/rules`, test tweaks

How does the packaging work? — The Debian Rust teams' workflow II

```
05:00:13 werdahias@yggsdrasil ~ → nala show  librust-async-tls-dev -a
Package: librust-async-tls-dev
Version: 0.12.0-2
Architecture: amd64
Installed: no
Priority: optional
Essential: no
Section: rust
Source: rust-async-tls
Origin: Debian
Maintainer: Debian Rust Maintainers <pkg-rust-maintainers@alioth-lists.debian.net
>
Installed-Size: 113 KB
Provides:
  librust-async-tls+default-dev
  librust-async-tls+early-data-dev
  librust-async-tls+server-dev
  librust-async-tls+webpki-dev
  librust-async-tls-0+default-dev
  librust-async-tls-0+early-data-dev
  librust-async-tls-0+server-dev
  librust-async-tls-0+webpki-dev
  librust-async-tls-0-dev
  librust-async-tls-0.12+default-dev
  librust-async-tls-0.12+early-data-dev
  librust-async-tls-0.12+server-dev
  librust-async-tls-0.12+webpki-dev
  librust-async-tls-0.12-dev
  librust-async-tls-0.12.0+default-dev
  librust-async-tls-0.12.0+early-data-dev
  librust-async-tls-0.12.0+server-dev
  librust-async-tls-0.12.0+webpki-dev
  librust-async-tls-0.12.0-dev
Depends:
  librust-futures-core-0.3+default-dev (>= 0.3.5-~~)
  librust-futures-io-0.3+default-dev (>= 0.3.5-~~)
  librust-rustls-0.21+default-dev
  librust-rustls-pemfile-1+default-dev
  librust-webpki-0.22+default-dev
Replaces: librust-async-tls-dev
Breaks: librust-async-tls-dev (!= 0.12.0-2)
Homepage: https://github.com/async-std/async-tls
Download-Size: 23 KB
APT-Sources: http://ftp.de.debian.org/debian/ unstable/main amd64 Packages
Description: Asynchronous TLS/SSL streams using Rustls - Rust source code
 This package contains the source for the Rust async-tls crate, packaged by
 debcargo for use with cargo and dh-cargo.
```

What is `gtk-rs` ? - The base for applications

- bindings for the C GTK libraries
- building on top of glib

What is `gtk-rs` ? - The base for applications

- bindings for the C GTK libraries
- building on top of glib
- usually one low-level -sys library for direct C calls
- . . . and a high-level library for safe API calls

What is `gtk-rs` ? - The base for applications

- bindings for the C GTK libraries
- building on top of glib
- usually one low-level -sys library for direct C calls
- . . . and a high-level library for safe API calls
- Example: libadwaita-sys and libadwaita

What is `gtk-rs` ? - The base for applications

- bindings for the C GTK libraries
- building on top of glib
- usually one low-level -sys library for direct C calls
- . . . and a high-level library for safe API calls
- Example: libadwaita-sys and libadwaita
- most applications use GTK4 + libadwaita (GTK3-rs slowly being deprecated)

How do (GTK-) Rust applications get packaged? — Offline building I

- "translating" `Cargo.toml` dependencies into debian ones
- `debian/control` and `debian/rules`: rust-specific
- the rest: regular Debian packaging workflow: `debian/watch`, `debian/copyright` etc.
- currently: statically linked

How do (GTK-) Rust applications get packaged? — Offline building II

```
Cargo.lock  Cargo.toml
05:19:27 werdahias@yggsdrasil ~ → cat obfuscate-0.0.9/Cargo.toml
[package]
name = "obfuscate"
version = "0.0.9"
authors = ["Bilal Elmoussaoui <bil.elmoussaoui@gmail.com>"]
edition = "2021"

[dependencies]
gtk = { package = "gtk4", version = "0.7", features = ["gnome_45"] }
log = "0.4"
gettext-rs = { version = "0.7", features = ["gettext-system"] }
pretty_env_logger = "0.5"
anyhow = "1.0"
adw = {package = "libadwaita", version = "0.5", features  = ["v1_4"]}
```

How do (GTK-) Rust applications get packaged? — Offline building III

```
05:14:49 werdahias@yggsdrasil ~ → cat obfuscate-wip/debian/control
Source: obfuscate
Section: misc
Priority: optional
Maintainer: Debian GNOME Maintainers <pkg-gnome-maintainers@lists.alioth.debian.or
Uploaders: Matthias Geiger <werdahias@riseup.net>
Build-Depends:  debhelper-compat (= 13),
                libgtk-4-dev,
                cargo:native,
                rustc:native,
                libadwaita-1-dev,
                libglib2.0-dev,
                gobject-introspection,
                libgdk-pixbuf-2.0-dev,
                meson (>= 0.53.0),
                desktop-file-utils,
                libstd-rust-dev,
                librust-anyhow-1+default-dev,
                librust-gettext-rs-0.7+default-dev,
                librust-gettext-rs-0.7+gettext-system-dev,
                librust-gtk4-0.7+default-dev,
                librust-gtk4-0.7+gnome-45-dev,
                librust-libadwaita-0.5+default-dev,
                librust-libadwaita-0.5+v1-4-dev,
                librust-log-0.4+default-dev,
                librust-pretty-env-logger-0.5+default-dev,
Standards-Version: 4.6.2
Homepage: https://gitlab.gnome.org/World/obfuscate
Vcs-Browser: https://salsa.debian.org/gnome-team/obfuscate
Vcs-Git: https://salsa.debian.org/gnome-team/obfuscate.git
Rules-Requires-Root: no

Package: obfuscate
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: Censor private information.
 Obfuscate lets you redact your private information from any image.
```

How do (GTK-) Rust applications get packaged? — Offline building

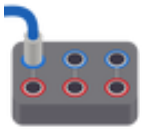
```
05:25:12 werdahias@yggsdrasil ~ → cat obfuscate-wip/debian/rules
#!/usr/bin/make -f

include /usr/share/dpkg/pkg-info.mk
include /usr/share/dpkg/architecture.mk
include /usr/share/dpkg/buildflags.mk
include /usr/share/rustc/architecture.mk
export CFLAGS CXXFLAGS CPPFLAGS LDFLAGS
export DEB_HOST_RUST_TYPE DEB_HOST_GNU_TYPE
export PATH := /usr/share/cargo/bin:$(PATH)
export CARGO=/usr/share/cargo/bin/cargo
export CARGO_HOME=$(CURDIR)/debian/cargo_home
export DEB_CARGO_CRATE=$(DEB_SOURCE)_$(DEB_VERSION_UPSTREAM)
export DEB_BUILD_MAINT_OPTIONS=hardening=+bindnow

%:
	dh $@

override_dh_auto_clean:
	dh_auto_clean
	rm -rf debian/cargo_registry

override_dh_auto_configure:
	 $(CARGO) prepare-debian debian/cargo_registry --link-from-system
	rm -f Cargo.lock
	dh_auto_configure
```

Why? — Advantages of Debian-native rust packages I

- wide range of architectures supported (amd64, arm (64, el, hf), i386, mips64el, riscv64, s390x)
- binary package does not depend on any runtime
- architecture- and size-optimized packages
- contributing to the wider linux ecosystem (e.g. mobile)
- reproducible build: offline and solely with debian tooling

Why? — Advantages of Debian-native rust packages II

Currently in debian:




Work in progress:




. . . and more to come !

What have we learned? — Summary

- **Rust ecosystem**

What have we learned? — Summary

- **Rust ecosystem**
- **Debian rust tooling**

What have we learned? — Summary

- **Rust ecosystem**
- **Debian rust tooling**
- **Debian-specific tweaks**

What have we learned? — Summary

- **Rust ecosystem**
- **Debian rust tooling**
- **Debian-specific tweaks**
- **The GTK-rs stack**

What have we learned? — Summary

- **Rust ecosystem**
- **Debian rust tooling**
- **Debian-specific tweaks**
- **The GTK-rs stack**
- **packaging of applications**