# PACKAGING RUST

## FOR DEBIAN, BUT ALSO OTHERS TOO

Angus Lees <gus@inodes.org>

## THE RUST TOOLCHAIN

- `rustc` and `rustdoc` (from rust-lang.org)
- `cargo`
- Libraries (crates) - written in Rust
- The application you actually wanted - written in Rust

## PACKAGING 101

### A BRIEF INTRODUCTION

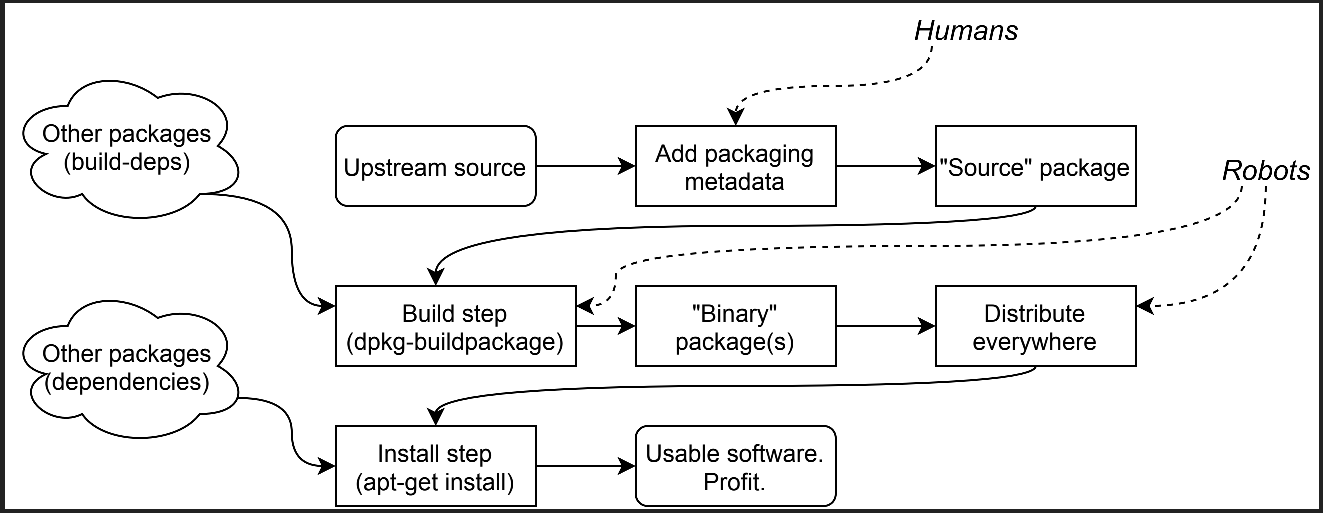
**Goal:** Create some metadata so the packaging system knows how to a) bundle up the software and b) make it easily installable.

Most distros[1] separate the *"build"* and the *"install"* steps so everyone can re-use the same generated artifacts.

[1] Notably not Gentoo or other "source-based" distros.

## PACKAGING 101

### THE PROCESS



## PACKAGING 101

### THE RULES

- License(s) must meet Debian Free Software Guidelines
- Must not use the network during build
- "Vendoring" sources is bad
  - hides them from security-team
  - duplication wastes resources
- Must be able to be rebuilt using packages in the archive
- Must be built on native architecture (no cross-compiling)
  - These last two are temporarily ignored during "bootstrapping" but result doesn't go into archive

## RUSTC AND FRIENDS

- This is in pretty good shape right now (for amd64)
- Looks a lot like a regular upstream project
  - Source `.tar.gz` releases, signed
  - `rustc` requires `rustc` to build, but this isn't unique
- Already in Debian *unstable*: https://packages.debian.org/sid/rustc
- Debian packaging maintained by a small team through http://anonscm.debian.org/cgit/pkg-rust/rust.git/
- Todo: cross compiling, easier bootstrap

## CIRCULAR BUILD-DEP ON RUSTC

- Currently handled by bundling the pre-built `rustc` stage0 blob with packaging metadata
- Works, but not great:
  - Large opaque binary makes people uneasy
  - Won't scale to many architectures (sheer size)
- Ideal future: Build `rustc` from itself
  - First architecture from pre-built blob
  - All other architectures cross-compiled
  - Future versions from existing rustc package
  - *Lots of blockers to address first*

## DIFFERENCES VS `MAKE INSTALL`

- Separate binary packages produced:
  - `rustc`
  - `rust-gdb`, `rust-lldb`
  - `rust-doc`
  - `libstd-rust-dev`
  - `libstd-rust-1bf6e69c`
- Split mostly to support (future) cross-compilation
  - `libstd-rust-dev` for target arch
  - `libstd-rust-xxx` can be co-installed for each arch

## DIFFERENCES VS `MAKE INSTALL`

- Run-time dylibs (`libstd-rust-xxx`) installed into regular `ld.so` path: `/usr/lib/x86_64-linux-gnu/lib*.so`
- Compile-time dylibs/rlibs (`libstd-rust-dev`) installed into: `/usr/lib/rustlib/x86_64-unknown-linux-gnu/lib/lib*.{so,rlib}`
  - dylibs (`*.so`) are symlinks back to run-time dylibs

## PATCHES APPLIED

- `src/llvm/*` removed (not needed)
- `jquery` source added
- `rust-{gdb,lldb}` scripts rewritten to hardcode paths
- `configure`/`Makefile` patch to pass `CFLAGS`/`LDFLAGS` down to build commands
  - `rustc`/`rustdoc` executables linked with `-Wl,-z,relro`
- `rustc` patch to add `-Wl,-soname=`*filename* when linking
- Documentation post-processed to use local icon/logos

## `RUSTC` OUSTANDING ISSUES

- Are we allowed to call it `rustc`?
- Only amd64,i386 architectures at this time
- "i386" arch package doesn't work on pentium (but does work on i686)
- Cross compilation not actually possible yet
  - mostly because LLVM packages aren't ready

## CARGO

- Packaging is a bit crude, but works
- Already in Debian *unstable*: https://packages.debian.org/sid/cargo

## `CARGO` PACKAGE BUILD PROCESS

- Crate dependencies bundled and shipped along with cargo source package
- A snapshot of crates.io-index is bundled and shipped along with cargo source package
- Uses a python script (from Bitrig) to build stage0 cargo (without using cargo)
- Generates `.cargo/config` to point to bundled registry and crates
- Creates a fake temporary git repo for index
- Points `CARGO_HOME` at an empty directory
- Finally run regular cargo `configure`/`make`

## PATCHES APPLIED

- Relax `missing_docs` lint in `aho-corasick`
- Fix relative paths in numerous bundled `Cargo.toml`s
- Remove cargo's `dev-dependencies` to prevent unnecessary attempted download

## CARGO-USING LIBRARIES

- Some early exploratory work, but mostly ideas so far
- Probably looks like Debian go-lang packages:
  - Library source installed into a central directory
  - Application builds pick up source from there
- Lots of issues still being worked on. See my recent post in the "Perfecting Rust Packaging - The Plan" thread on internals.rust-lang.org
  - Eg: Packaging from crates.io vs upstream repos
  - Overriding crate paths vs overriding cargo index

## CARGO-USING LIBRARIES (DYLIBS)

- *Can* support dylibs using tight package dependencies on `librust-xxx`
  - Need to be rebuilt following every compiler release
  - Will only do this if forced (compiler plugins?)

## CARGO-USING APPLICATIONS

- Once the libraries are solved, this should be easy!
- Run `cargo build --release`, copy the executable into the right directory
- Result will be an executable with no run-time dependencies on Rust crates (may require non-Rust libs)
- Need to be rebuilt following a Rust library security update

## THE LAST SLIDE

- pkg-rust-maintainers@lists.alioth.debian.org
- https://wiki.debian.org/Teams/RustPackaging
- #debian-rust on OFTC IRC network
- Packaging git repos: http://anonscm.debian.org/cgit/pkg-rust/
  - Applied patches are in `*/debian/patches/`
- Questions?