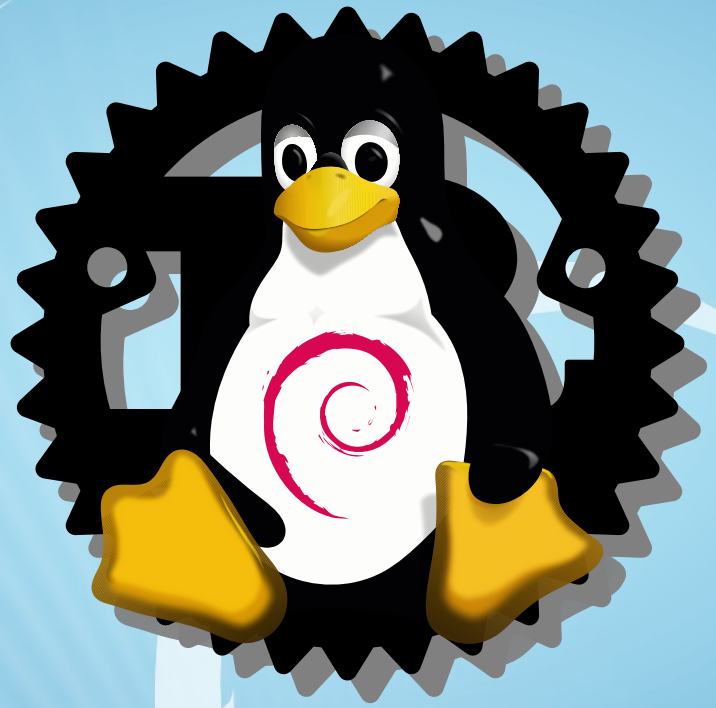
# Enabling Rust for Linux in Debian



Ben Hutchings · Kangrejos 2025 · Oviedo, Spain


# Ben Hutchings

- Doing kernel development for various employers and customers since 2008
- Member of Debian kernel team since 2009, working on configuration, build system, integration, backporting, review...
- Former stable branch maintainer (3.2, 3.16)
- Frustrated C programmer
- Rust newbie

# Debian project

- A community project:
  - Funded by donations of money and resources
  - Contributors mostly volunteers, but may be paid by outside organisation
- A constitutional democracy:
  - Members take part in annual leadership elections and general resolutions
  - Leader delegates specific powers and responsibilities
  - Technical committee rules on disputes
- But mostly runs on do-ocracy:

**The people who do the work, decide how to do it**

# Debian distribution

- Developers usually upload new source package versions to “unstable”
- Packages in unstable migrate into “testing” after several days, depending on QA results, dependencies, closeness to release
- Every ~2 years testing becomes the new stable release
  - Testing migration becomes progressively stricter, starting with a freeze on new toolchain versions
  - Most recently: Debian 13 “trixie”, 9 August 2025
- Each stable release supported for 5 years on most popular architectures
  - Security updates made every few days
  - Less urgent fixes aggregated into point releases every 2-4 months for first 3 years
  - No new upstream feature releases allowed, with rare exceptions

# Debian architecture support

1. Release architectures
   - included in stable releases; supported for 3+ years
   - most recently: Arm (v5, v7, v8), IBM Z, POWER LE, RISC-V 64, x86 (-32 and -64)
   - Arm (v7, v8) and x86 supported for 5 years (LTS)
2. Main archive
   - fully hosted on project infrastructure; expected to move to tier 1 or 3
   - currently: MIPS64 LE
3. Ports
   - hosted on mix of project and individual developer-hosted infrastructure
   - not expected to meet release standards
   - currently: Alpha, LoongArch 64, PA-RISC, Motorola 680x0, PowerPC (-32 and -64) BE, Super-H 4, SPARC 64, x86 x32, x86 with Hurd kernel

# Debian package builds

- Official packages are built *natively* for all supported architectures, on project infrastructure
  - Some upstreams do not support cross-building
  - Build-time tests cannot be run when cross-building
  - Packages *may* support cross-building for ease of development and to enable bootstrapping
- Official builds are **isolated**, and we aim for **reproducible** builds:
  - Each source package is built in a container with only “essential”, “build-essential”, and its declared build-dependencies installed
  - Container has no network access during the build
  - ➢ All build-dependencies must also be packaged or vendored
- GNU toolchain used for almost everything including kernel

# Rust in Debian (1)

- Debian Rust team does most of the Rust packaging work:
  - Defines a policy for packaging Rust crates
  - Develops tools to (mostly) automate the policy and create a local Cargo registry from installed packages
  - Maintains packages of Rust compiler, Cargo, and many crates (~3000 source packages in unstable)
- Other developers can also package Rust software, but are expected to follow the same policy
- Due to the unstable ABI, “binary” packages for library crates actually contain source code
- Library crates tested at build time with 0/1/all features enabled

# Rust in Debian (2)

- Compiler (`rustc` package) usually close to upstream in unstable, but subject to toolchain freeze ~6 months before a stable release
- Backported compiler (`rustc-web` package) provided in stable for building few applications that need to move to new upstream versions
- Choice of crates to package is mostly driven by applications and their dependencies
  - Kernel could be one of those “applications” if it doesn’t vendor required crates
  - Rust developers expected to install library crates with Cargo, not APT

# Security woes of static linking

- Debian policy: don’t duplicate source code; use shared libraries
  - ➢ Allows most security issues to be addressed with a single source upload and rebuild
- Rust and Go applications tend to statically link large numbers of libraries
  - ➢ Fixing security issue in a library requires rebuilding all reverse-dependencies, resulting in long build queues for security updates
- Debian infrastructure for mass-rebuilding reverse-dependencies was designed for unstable, not stable security updates
- ➢ Rust and Go security updates may be deferred to a point release
- This does not prevent leaf packages getting their own security updates
  - Example: Firefox ESR (but it relies on vendoring)

# Linux kernel in Debian (1)

- Debian kernel team packages Linux kernel and closely related software
- Small patch set; we aim for “upstream first” so we don’t need to forward-port
- Each Debian stable release follows an upstream “longterm” stable branch:
  - Debian 12 “bookworm” follows Linux 6.1.y
  - Debian 13 “trixie” follows Linux 6.12.y
  - Debian 14 “forky” will probably follow 2026 longterm branch
- testing and unstable currently have 6.16.y, but will keep moving to new upstream versions until freezing in preparation for “forky”
- Each stable release also gets backport of kernel from next release or unstable
  - Backports are built using the toolchain and other build-deps from the older release

# Linux kernel in Debian (2)

- Out-of-tree module packages contain source to be built on end user systems
- Current source package supports this by building multiple binary packages:
  - `linux-kbuild-`*VERSION*: The kernel build system and tools, including conf, objtool, etc.
    - Some tools built multiple times for different target architectures, with wrappers to select the right version
  - `linux-headers-`*VERSION*-common: Static kernel headers
  - `linux-headers-`*VERSION-FLAVOUR*: `.config`, generated kernel headers, etc. for a single kernel “flavour”
  - Later OOT module builds may be native or cross, using build architecture’s `linux-kbuild` and host architecture’s `linux-headers`

# Enabling Rust for Linux in Debian

- Debian Bananas team maintains unofficial package based on Asahi kernel, with RfL enabled as dependency of graphics drivers
- I have an open merge request to enable RfL in the official kernel package, including support for building out-of-tree modules
- Open problem: support for cross-building and OOT modules
  - RfL builds `.rmeta` and `.so` files that depend on both the *build* architecture and `.config`, which doesn't fit into the current scheme
  - ➢ May need to rebuild these files during OOT module build, but how?
  - ➢ Changing build architecture between kernel and OOT module build should not affect kernel ABI; is this true for rustc?
  - ➢ Maybe OOT module support should be deferred to later?

Questions?

# Credits & License

- Content by Ben Hutchings
  www.decadent.org.uk/ben/talks/
  License: GPL-2+
- Original OpenOffice.org template by Raphaël Hertzog
  raphaelhertzog.com/go/ooo-template
  License: GPL-2+
- Background based on “Serenity” theme by Edward Padilla
  wiki.debian.org/DebianArt/Themes/serenity
  License: GPL-2