Making Old Things New

Reuben Thomas

Old Things

- GNU has many “old-fashioned” packages which are central to its operation:
  - coreutils, grep, diffutils…
  - not to mention bash, glibc, Linux

Legacy code in maintenance mode?

- Working on these programs is not seen as fun or cool. A pity:
  - They are “merely” maintained
  - The lower levels of the stack (other than Linux & gcc) are not seen as a good place to innovate
  - Programming with GNU tools is widely seen as dull, unproductive & old-fashioned
    - Despite many improvements, this is still truth in this perception
- Is it really worth it?

Plan of this talk

- Explain why it *is* still worth working on “old” stuff
- Showcase recent advances
  - Look at some projects that use them
- See what still sucks...
- ...and make suggestions to fix it
- Point to some potential areas for innovation

Fundaments, not monuments

- Isn’t this all legacy stuff?
- Many developers still work at the command line
- CLI still the most flexible & universal interface
- To improve a system, best start at the bottom
- Mature ≠ Dead
  - POSIX & ISO C are still developing

GNU is falling behind (the HURD)

- GNU is developing too (more on that later)
- But too slowly: enough *maintainers...*
  - Core reasonably well-maintained & polished
- ...but not enough *developers*
  - Most developers are working on desktop, web & networking projects
  - GCC is a test-bed for new ideas; coreutils is not
- Lack of work on the HURD is symptomatic (?)

Some new old stuff

- A personal selection of things I’ve used recently:
  - gnulib
  - gcc
  - autoconf-archive
  - ccache
  - Emacs
- Don’t forget automake, autoconf, libtool, gcc &c.

gnulib: turbocharge your autotools

- One-stop shop for compatibility
- Data structures and other general-purpose code
- Incubator for
  - patches
  - new GNU APIs
- libposix
- Build system magic
  - from linting to distribution

You have to fit the turbo yourself

- gnulib-tool is excellent, but unusual
  - Imagine a world in which everything's like autoconf
- libposix is not yet released
  - Source & binary bloat
- No signposting of maturity/standardization
  - Always use the latest version, for good or ill
- Up to now, adoption of gnulib largely driven by its authors

gcc

- (Almost-)complete C99 support (finally!)
- C0x features appearing
- C++0x, Obj-C 2 partly supported
- Ongoing improvements to FORTRAN & Ada
- “Invisible” improvements: diagnostics, optimization
- Go (more on that later)

autoconf-archive

- 100s of macros
- Common (and uncommon) configuration tasks
- Detect dozens of languages & libraries
  - When one needs to know versions, optional bits &c.
- A *de facto* repository of best practice
- If you write more than 6 lines for a task, submit a macro to autoconf-archive!

ccache

- Eliminates repeat compilations for C, C++, Objective C, FORTRAN
- Transparent
- For developers, mostly useful for fiddling with configure settings
  - Don’t use it to work around build system bugs!

Emacs

- ✔ flymake
- ✔ whitespace-mode
- ✗ CEDET
- ✗ nXhtml

Valgrind

- Surely we’re all using it by now?
- Use it to run test suites
  - gnulib’s parallel-tests target supports using Valgrind

Example 1: grep ≥2.6

- gnulibized
- Code removed
- Files merged with other projects
- Build system updated
- Why the lack of excitement?
  - As Jim Meyering said: “it’s *grep*!”
- But most of the work done by gnulib folk (Meyering, Bonzini, Blake)

Example 2: Zile ≥ 2.3

- Zile is a lightweight Emacs subset
- Provides the features & commands most often used
- For quick editing an experienced Emacs user should rarely get “muscle memory shock”
- Typically about 200Kb binary (~80Kb of gnulib!)

Zile 2.3: gnulib

- Used gnulib for portability
- Added test suite
  - First version with DejaGnu
    - But expect had timing problems
  - Second version with Lisp scripts
    - Extra features needed, but test suite runs on Emacs too
- 2 kLOC removed (~20% of C)
- Emphasis on behaving exactly like Emacs
  - Made debugging & design decisions easier!

Zile 2.4: C99, POSIX-1.2008, GC

- C99
  - Declarations anywhere → shorter, more readable code
- POSIX-1.2008
  - No need for non-standard APIs
- GC
  - Many code paths simplified by not having to manage memory
- 1 kLOC removed (~15% of C)

Spreading the word

- Current improvements are absorbed by maintenance
- Wouldn’t it be great if more people joined in to push things forward?
- Sometimes change helps because it catches the imagination

Zile 2.5: Lua

- ✔ Code simplified
  - 80% of size of C
  - Considerable room for further simplification (want at least 50%)
- ✔ No compilation
- ✗ Some classes of bug easier to introduce
  - No static checks
- ✔ An inducement to improve test coverage!

Lua

- MIT license
- Dynamic
  - Compact code
  - No compilation
- Familiar (Pascal/Python-like syntax)
- Small (14 kLOC)
- Mature (>15 years, i.e. about the same as C89)
- First-class functions, closures, co-routines
- Designed to integrate with C & C++
- Highly portable (written in intersection of C++ & C89)
- One of the fastest dynamic languages (LuaJIT)

Writing GNU software in Lua

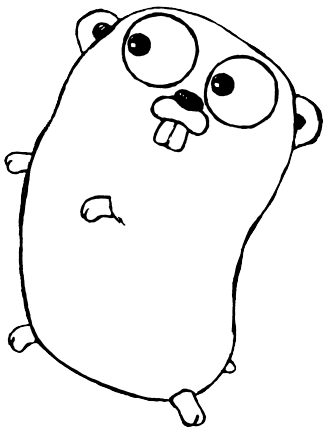
- If in doubt, ship Lua with your code
- Can do all we want (embedding or standalone) with autotools
- Libraries: luaposix, lcurses, lrexlib
  - ✗ Incomplete
  - ✗ Inconvenient to install
  - ✔ ctypesgen-json for dynamic binding
- Anyone else want to give it a go?

What still sucks

- autoconf
  - Slow, fragile, messy
  - Un-reimplementable
  - Get rid of it by removing need for it to the point where a much simpler replacement suffices
- automake
  - Compiled down to POSIX Make
- Out of date system dependencies
  - POSIX sh

Thrill seeking

- An aside: Go
  - Seriously exciting: a GC'd systems language with a lightweight type system, concurrency (“goroutines”) & module system, instant compilation, and performance within 10-20% of C.
- **Instant compilation!**



Fix 1: More up-to-date requirements

- C99
- POSIX-1.2008 (via gnulib)
  - Install gnulib (libposix) as system veneer
    - Force portability improvements to be shared between apps
    - Cut build times and source tarball sizes
- GNU Make
  - quagmire
- Perl
  - Please, no more shell scripts!
    - Quoting is to the shell as memory management is to C

Fix 2: Faster builds

- ccache
  - Debian: install symlinks in /usr/bin, not /usr/lib/ccache!
- autoconf cache
  - On by default
  - Multiple configure runs in one cache?
- Pre-compiled headers
  - Shipped for base GNU, GNU+glib, GNU+glib+gtk

Fix 3: Smaller tool surface

- Assume high-level tools
  - automake
  - autoconf
  - libtool
  - Valgrind
- Make it unnecessary to learn about & switch on each tool one by one
- *See also* Emacs

Fix 4: More automation

- More regression/unit tests
  - make distcheck
- More nix
- More analysis
  - make syntax-check
- More tools
  - Executable README-release, please!
- Help convince conservative maintainers that changes won’t break mature code

Fix 5: Better dev environments

- Anjuta (for IDE fans)
- Emacs
  - CEDET
  - Flymake
  - nXhtml
  - gdb for console apps
- Instead of minimal deps, easily install GNU
  - à la brew
  - Helps GNU more generally

Fix 6: Think globally, act globally

- Encourage code to be injected lower down, not bubble up from individual programs
- Speed up integration of mature code
  - autoconf-archive → autoconf, gnulib → glibc
  - Documented processes
- Transparent decompression
  - patch to grep
  - zutils
  - Why not a global approach?

Synergy

- The effect of better tools is cumulative
- The longer the lever, the more leverage you get
- These truisms are not emphasised enough in GNU

Polish the past, glimpse the future

- This is not just shining the family silver…
- …it’s rubbing the genie’s lamp!
- Go back to basics
  - Ubiquitous compression
  - Extend the file system
    - Network access (httpfs &c.)
    - Structured data introspection (physicsfs &c.)
  - XML coreutils (xml-coreutils &c.)
  - Next-gen terminal (ZUI, Cobra Command Tool, Subtext, numscipy, mc)

Why do we do it?

- Have fun
- Improve our toys
- Teach others
- Make the world a better place
- ...

- Programming is communication
- We communicate our values